Unifying Reasoning and Planning through Energy Minimization

Adrian Rodriguez[1] **Angelica Kim**[2] **Yunhui Guo**[1] **Yilun Du**[2]
[1] The University of Texas at Dallas [2] Harvard University

Abstract

Neural networks often struggle to solve reasoning and planning problems more complex than those encountered during training. Energy-based models offer a natural approach to this challenge by learning objectives that can be optimized with additional computation. However, existing methods rely on specialized training and inference procedures, from differentiating through iterative optimization to annealing across noise-dependent energy landscapes. We introduce *Energy Minimization Search* (EMS), a unified framework for reasoning and planning through a shared training objective and time-invariant energy minimization. EMS combines Equilibrium Matching with contrastive supervision to learn an energy function for each task, using the energy values to evaluate candidates and energy gradients to guide refinement. At inference time, we search on this fixed landscape, allocating additional computation to more refinement steps or candidate solutions. Across graph coloring, N-Queens, 3-SAT, crosswords, path finding, and maze navigation, EMS achieves competitive or improved solution quality relative to baselines, with substantially fewer function evaluations than the diffusion-based EBM reasoning baselines. Project website: https://energy-based-model.github.io/ems/.

1 Introduction

A central goal of artificial intelligence is to build systems that solve problems they have never seen before [8], including reasoning problems with more constraints and planning problems with longer horizons than those encountered during training. Reasoning is often framed as constraint satisfaction over discrete assignments [13], while planning involves trajectories or sequential decisions [32]. Despite differences in their representations and models, both admit a common search structure: proposing candidate solutions, evaluating their compatibility with problem conditions, and refining them [46]. This shared structure motivates learning objectives that guide search across both domains.

Energy-based models (EBMs) [33, 14, 25] provide a natural mechanism for this view. An EBM is trained to assign lower energy to valid solutions than to incompatible candidates, providing a learned compatibility score. Gradients of the same energy function can then refine candidate solutions toward lower energy, turning solution generation into optimization-based search. When a problem decomposes into local constraints, their energies can be summed to guide search over the complete solution [42]. Thus, both domains admit optimization in a learned energy landscape.

Prior work applies energy-based modeling to reasoning [16, 17, 42] and planning [20, 15, 11], using different parameterizations, training objectives, and search procedures. Recent diffusion-based EBMs learn time-dependent landscapes conditioned on noise level. Inference traverses these landscapes according to a noise schedule, sometimes performing multiple refinement updates within each landscape [17, 42]. Although annealing can facilitate search, it requires coordinating refinement across changing objectives. This raises the question of whether a shared training objective can support effective reasoning and planning by learning energy functions that remain fixed throughout search.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

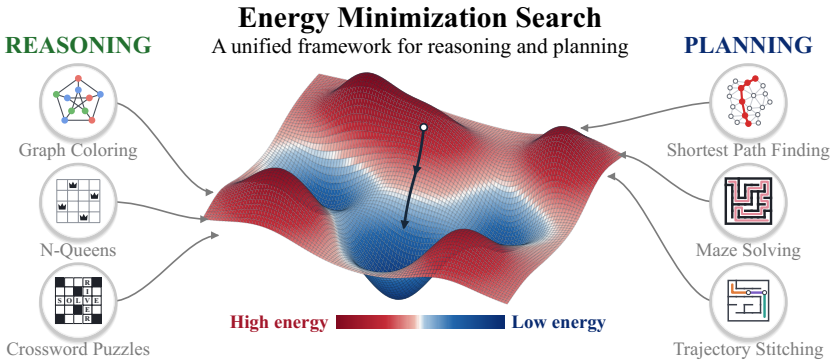

Figure 1: **Unifying reasoning and planning through energy minimization.** EMS combines Equilibrium Matching and contrastive supervision to learn a time-independent energy function for each task, then refines candidate assignments or trajectories through gradient-based search.

We introduce *Energy Minimization Search* (EMS), a unified framework for reasoning and planning through time-invariant energy minimization (Figure 1). EMS combines Equilibrium Matching [57] with contrastive supervision. Equilibrium Matching supervises gradients that guide invalid candidates toward valid solutions, while contrastive supervision encourages valid solutions to have lower energy than incompatible alternatives. The same training objective and energy minimization formulation apply across both reasoning and planning. At inference time, EMS uses gradient-based search to refine candidates on a fixed energy landscape. Additional computation can be allocated to more refinement steps or more candidates during inference.

Our contributions are threefold. First, we present EMS as a shared framework for reasoning and planning through a common training objective and time-independent energy minimization. Second, we evaluate it on graph coloring, N-Queens, 3-SAT, crosswords, path finding, and maze navigation, finding competitive or improved solution quality relative to the evaluated baselines, with substantially fewer function evaluations than the evaluated diffusion-based EBMs on all four reasoning tasks. Third, we analyze the effects of contrastive supervision, search methods, and inference budgets, and characterize the remaining limitations of compositional long-horizon planning.

2 Related Work

Diffusion for Reasoning and Planning. Diffusion models [26, 48] generate candidates through iterative denoising, with applications to reasoning [17, 60], trajectory planning [29, 2], motion and task planning [9, 38, 58, 18], and hierarchical planning [35, 10]. CompDiffuser stitches short trajectories through diffusion composition [39], while Energy-based Compositional Diffusion Planning constructs a global energy from local denoiser potentials and boundary interactions [51]. These approaches use noise-conditioned denoising or energy refinement during inference. EMS instead applies time-independent energy minimization to both reasoning and planning.

Reasoning as Optimization and Iterative Computation. Neural reasoning systems incorporate structured computation through differentiable satisfiability solvers [56], theorem proving [44, 41], probabilistic logic [40], and optimization layers [3, 53]. Other approaches use neural programs [23, 43, 62], adaptive computation [22, 7], or recurrent refinement [55, 30, 4, 24]. EMS expresses refinement through gradients of a learned scalar energy, with the search method and inference budget chosen separately from the trained model.

Reasoning and Planning as Energy Minimization. EBMs provide learned objectives for reasoning [16, 21] and planning [15, 20, 11]. IREM and EBT differentiate through iterative solution updates, while IRED uses diffusion-based gradient and energy supervision [17]. Compositional energy-based reasoning sums learned constraint energies at inference [42], and CCEM trains input-convex energies with an unrolled projected solver [45]. Time-independent energy learning [49] predates EMS, including contrastive divergence [25], score matching [28, 50], and Energy Matching [5, 6]. EMS adopts the explicit energy formulation of Equilibrium Matching [57], adding contrastive supervision for conditional reasoning and planning without an inner solution refinement loop during training.

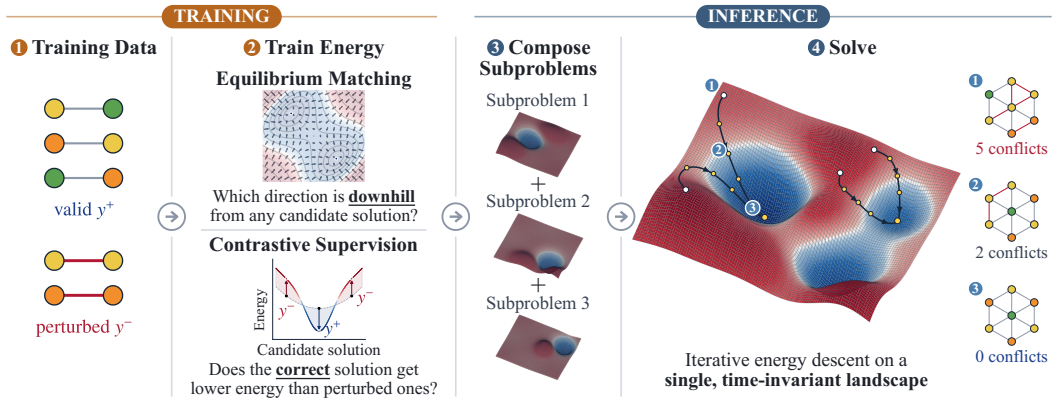

Figure 2: **Overview of Energy Minimization Search (EMS)**, illustrated on graph coloring. **(1)** Training data pair valid solutions y^+ with perturbed negatives y^-. **(2)** Equilibrium Matching and contrastive supervision train a single time-invariant energy E_θ whose gradients point toward valid solutions, which receive lower energy than perturbed ones. **(3)** At inference, subproblem energies are summed into one global objective. **(4)** Gradient-based search on this fixed landscape iteratively refines a candidate, here from 5 conflicts to a valid coloring.

3 Energy Minimization Search

3.1 A Unified Energy Formulation

We formulate reasoning and planning as optimization over a learned energy landscape. The energy function evaluates the compatibility of a candidate solution with the problem conditions, while its gradients guide iterative refinement. Figure 2 provides an overview of the framework.

Let $\mathbf{x}$ denote the conditions of a problem instance, such as logical constraints or an environment with start and goal configurations, and let $\mathbf{y}$ denote a candidate assignment or trajectory. We train an energy function $E_\theta : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to \mathbb{R}$ to assign lower energy to valid solutions than to incompatible candidates [33]. Inference then corresponds to minimizing this energy with respect to the candidate solution:

$$\mathbf{y}^* \in \operatorname*{argmin}_{\mathbf{y}} E_\theta(\mathbf{x}, \mathbf{y}). \tag{1}$$

We approximate this minimization using the gradient-based search methods described in Section 3.3. For discrete tasks, we optimize continuous relaxations of candidate assignments and decode them into discrete solutions before evaluating metrics. Task encoding and decoding is described in Appendix B.

EMS uses the same training objective and energy-minimization formulation for reasoning and planning. We train a separate model for each task and keep its energy landscape fixed throughout inference. The representation, architecture, and search settings follow the corresponding benchmark. Appendix A.1 provides details on energy parameterization and training.

3.2 Learning a Static Energy Landscape

We train an energy function to both evaluate candidate solutions and guide their refinement. This requires assigning lower energy to valid solutions while learning gradients that direct candidates toward them. We address these two requirements jointly through Equilibrium Matching, which supervises refinement directions, and contrastive supervision, which distinguishes valid solutions from incompatible alternatives. Both objectives train the same static energy function.

Equilibrium Matching. We use the explicit energy formulation of Equilibrium Matching (EqM) [57] to learn gradients that guide corrupted candidates toward valid solutions. For a training pair $(\mathbf{x}, \mathbf{y})$, we sample Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and a coefficient $\gamma \sim \mathcal{U}(0, 1)$, then construct a corrupted candidate:

$$\mathbf{y}_\gamma = \gamma \mathbf{y} + (1 - \gamma)\boldsymbol{\epsilon}. \tag{2}$$

We train the energy function by minimizing

$$\mathcal{L}_{\text{EqM}} = \mathbb{E}\left[\left\|\nabla_{\mathbf{y}_\gamma} E_\theta(\mathbf{x}, \mathbf{y}_\gamma) - \lambda(\boldsymbol{\epsilon} - \mathbf{y})c(\gamma)\right\|_2^2\right]. \tag{3}$$

Algorithm 1 EMS Training Update

Require: Data $\mathcal{D}$, energy E_θ, λ, β, negative construction rule $\mathcal{P}$

Sample $(\mathbf{x}, \mathbf{y}^+) \sim \mathcal{D}$
Sample $\gamma \sim \mathcal{U}(0,1)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Construct $\{\mathbf{y}_j^-\} \leftarrow \mathcal{P}(\mathbf{x}, \mathbf{y}^+)$
Corrupt $\mathbf{y}^+$ and $\{\mathbf{y}_j^-\}$ with shared $\gamma, \boldsymbol{\epsilon}$
$\mathbf{g} \leftarrow \lambda(\boldsymbol{\epsilon} - \mathbf{y}^+)(1-\gamma)$
Compute $\mathcal{L}_{\text{EqM}}$ with target $\mathbf{g}$ using Eq. (3)
Compute $\mathcal{L}_{\text{CL}}$ using Eq. (4)
Update θ using $\nabla_\theta(\mathcal{L}_{\text{EqM}} + \beta\mathcal{L}_{\text{CL}})$
return θ

Algorithm 2 EMS Inference (Gradient Descent)

Require: Conditions $\mathbf{x}$, energy E_θ, particles P, steps K, step size η

Construct full-instance or composed E_{search}
for $p = 1, \ldots, P$ **do**
  Initialize $\mathbf{y}_p^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $k = 0, \ldots, K-1$ **do**
    $\mathbf{y}_p^{k+1} \leftarrow \mathbf{y}_p^k - \eta \nabla_{\mathbf{y}} E_{\text{search}}(\mathbf{x}, \mathbf{y}_p^k)$
  end for
end for
$p^* \leftarrow \operatorname{argmin}_p E_{\text{search}}(\mathbf{x}, \mathbf{y}_p^K)$
return $\text{Decode}(\mathbf{y}_{p^*}^K)$

Here, $\lambda > 0$ scales the target gradient. The target energy gradient points from the clean solution toward noise, so moving along its negative guides the corrupted candidate toward the solution. We set $c(\gamma) = 1 - \gamma$ so that the target gradient vanishes as γ approaches the clean solution endpoint. We use γ to construct the corrupted candidate and training target, but do not provide it to the energy model. The learned energy landscape therefore remains fixed throughout inference.

Contrastive Supervision. EqM supervises refinement directions but does not explicitly require valid solutions to have lower energy than nearby invalid candidates. We therefore add contrastive supervision [17, 42]. For each clean solution $\mathbf{y}^+ = \mathbf{y}$, we construct negative candidates $\{\mathbf{y}_j^-\}_{j=1}^J$ directly using task-specific perturbations or constraint violations. For example, we perturb an N-Queens row by removing its queen or adding queens at other positions.

We corrupt the positive and each negative using Eq. (2) with the same coefficient γ and noise $\boldsymbol{\epsilon}$, obtaining $\mathbf{y}_\gamma^+$ and $\{\mathbf{y}_{\gamma,j}^-\}_{j=1}^J$. We use the same corrupted positive in both the EqM and contrastive objectives. The contrastive loss encourages this positive to have lower energy than the corresponding negatives:

$$\mathcal{L}_{\text{CL}} = \mathbb{E}\left[-\log \frac{\exp[-E_\theta(\mathbf{x}, \mathbf{y}_\gamma^+)]}{\exp[-E_\theta(\mathbf{x}, \mathbf{y}_\gamma^+)] + \sum_{j=1}^J \exp[-E_\theta(\mathbf{x}, \mathbf{y}_{\gamma,j}^-)]}\right]. \tag{4}$$

We combine the two objectives as $\mathcal{L} = \mathcal{L}_{\text{EqM}} + \beta\mathcal{L}_{\text{CL}}$, where $\beta \geq 0$. Crosswords and full-instance PointMaze planning use EqM alone ($\beta = 0$). We select λ and β using validation instances and report task-specific perturbations and settings in Appendices B and C. Algorithm 1 summarizes training.

3.3 Composition and Search

An advantage of EBMs is that they separate the learned representation of a problem from the procedure used to solve it. This allows us to choose the inference algorithm and computational budget while keeping the learned energy function fixed. In EMS, we first construct an energy over the complete problem, either directly or by composing local energies. We then search this landscape using different refinement procedures, with additional computation allocated to more refinement steps or candidate solutions at inference time.

Full-Instance and Compositional Energies. When trained on complete assignments or trajectories, EMS learns an energy over the full solution, which we minimize directly at inference time. With local training examples, such as colored edges or satisfied clauses, we follow prior work [42] and construct a global energy compositionally by summing the local energies:

$$E_{\text{comp}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} E_\theta(\mathbf{x}_i, \mathbf{y}_i), \tag{5}$$

where $\mathbf{x}_i$ describes local component i and $\mathbf{y}_i$ contains the corresponding variables from the global candidate $\mathbf{y}$. A full-instance energy is the special case $N = 1$. Components can share variables, so each variable receives gradient contributions from all local energies in which it appears. Minimizing the sum therefore jointly refines assignments across overlapping constraints.

Composition allows the same local model to be reused across problems with different numbers and arrangements of constraints, provided the local input representations remain compatible. For graph

coloring, a model trained on valid color assignments to pairs of adjacent nodes can be applied to larger graphs by adding one energy term per edge. Both full-instance and compositional energies thus use the same learning objective and search formulation.

Search. We consider three search procedures on EMS, with different benefits for refinement and exploration. Each operates on the same fixed energy landscape, starting from P candidates searching in parallel and initialized independently from Gaussian noise. For brevity, the updates below use E_θ as the global energy for full-instance and compositional energies.

Gradient Descent (GD). GD provides deterministic refinement by following the negative energy gradient:

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \eta \nabla_{\mathbf{y}} E_\theta(\mathbf{x}, \mathbf{y}^k), \tag{6}$$

where $\eta > 0$ is the step size and k indexes refinement steps.

Unadjusted Langevin Algorithm (ULA). ULA adds independent Gaussian noise to each update, allowing candidates to explore beyond deterministic descent trajectories and help escape poor local minima:

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \eta \nabla_{\mathbf{y}} E_\theta(\mathbf{x}, \mathbf{y}^k) + \sqrt{2\eta}\, \boldsymbol{\xi}^k, \qquad \boldsymbol{\xi}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{7}$$

Parallel Energy Minimization (PEM). PEM [42] concentrates computation on candidates with lower energy. At each iteration, we resample P particles $\{\mathbf{y}_p^k\}_{p=1}^P$ with replacement using weights $w_p = \mathrm{softmax}_p(-E_\theta(\mathbf{x}, \mathbf{y}_p^k))$. We then refine the resampled particles $\widetilde{\mathbf{y}}_p^k$:

$$\mathbf{y}_p^{k+1} = \widetilde{\mathbf{y}}_p^k + \sigma \boldsymbol{\xi}_p^k - \eta \nabla_{\mathbf{y}} E_\theta(\mathbf{x}, \widetilde{\mathbf{y}}_p^k), \qquad \boldsymbol{\xi}_p^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{8}$$

where σ is the noise scale. Independent noise allows duplicated particles to explore different refinement trajectories.

After refinement, we select the candidate with the lowest energy and decode it into a solution. The search method, refinement steps, and candidate count can all be changed without retraining the energy function. Algorithm 2 summarizes GD inference. Additional search variants and task-specific comparisons appear in Appendix A.2 and Appendices B–C, respectively.

4 Experiments

We evaluate EMS on four compositional reasoning tasks, shortest path finding, continuous trajectory planning, and long-horizon planning by stitching short segments. We compare solution quality and model evaluation counts, and study the effects of contrastive supervision and search methods.

For comparisons with diffusion baselines, we retain their backbones without time conditioning and follow their training data, encoding, decoding, and evaluation protocols [17, 42, 2, 39]. We report model forward evaluations (NFE) per candidate, including refinement and scoring, and totals across candidates (Table 7, Appendix A.3). Task settings, baseline differences, and ablations appear in Appendices B and C. Shared method details appear in Appendix A.

4.1 Reasoning

4.1.1 Graph Coloring

Task and Setup. We first test whether energies learned from individual edges can guide search over an entire graph. Graph coloring assigns one of k colors to each node so that adjacent nodes receive different colors. Following Oarga and Du [42], we use the graph's chromatic number as k and measure the number of conflicting edges. EMS trains on valid colored edges and sums their energies over each test graph. We evaluate five graphs per distribution and compare against GCN [31], GAT [54], XLVIN [12], GNN-GCP [34], and DiffEBM [42]. EMS uses ULA with 128 candidates and 500 updates, matching DiffEBM's candidate count.

Quantitative Results. EMS achieves the lowest mean conflict count across all eight distributions (Table 1). It finds valid colorings for every evaluated Holme Kim (Small), Regular Expander (Small), and Complete graph. On Holme Kim (Large), it reduces mean conflicts from 59.0 for DiffEBM to 3.4. Paley graphs remain more difficult, with 20.6 mean conflicts for EMS compared with 34.8 for DiffEBM. These improvements use the same number of candidates: EMS requires 64,128 total

Distribution	GCN	GAT	XLVIN	GNN-GCP	DiffEBM	EMS
Erdos Renyi (Small)	46.80 ± 20.47	34.00 ± 11.55	25.00 ± 7.81	15.20 ± 4.32	8.60 ± 4.82	$\mathbf{0.20 \pm 0.40}$
Erdos Renyi (Large)	151.60 ± 12.09	130.20 ± 11.47	93.80 ± 31.12	53.80 ± 8.34	29.20 ± 8.05	$\mathbf{1.40 \pm 1.50}$
Holme Kim (Small)	74.00 ± 14.74	51.20 ± 11.47	29.00 ± 7.75	13.20 ± 6.65	10.60 ± 2.70	$\mathbf{0.00 \pm 0.00}$
Holme Kim (Large)	408.00 ± 26.40	253.20 ± 50.71	182.60 ± 24.73	55.20 ± 12.63	59.00 ± 3.74	$\mathbf{3.40 \pm 1.85}$
Regular Expander (Small)	87.60 ± 22.58	58.60 ± 12.44	29.00 ± 7.75	15.40 ± 6.65	11.00 ± 4.89	$\mathbf{0.00 \pm 0.00}$
Regular Expander (Large)	144.80 ± 6.90	118.80 ± 12.59	112.60 ± 10.97	141.60 ± 69.47	37.20 ± 4.71	$\mathbf{2.00 \pm 0.89}$
Paley	285.00 ± 117.70	239.20 ± 159.43	151.80 ± 92.13	91.20 ± 63.14	34.80 ± 20.27	$\mathbf{20.60 \pm 16.99}$
Complete	46.00 ± 15.04	46.00 ± 15.04	34.80 ± 16.42	30.00 ± 2.54	3.40 ± 1.14	$\mathbf{0.00 \pm 0.00}$

Table 1: **Graph coloring.** Number of conflicting edges (lower is better). EMS reports mean $\pm$ standard deviation across five graphs per distribution. Baseline values are reproduced from Oarga and Du [42]. Both EMS and DiffEBM use 128 candidates.

evaluations per graph, compared with 64,320,128 for DiffEBM with PEM, approximately a 1,000$\times$ reduction, showcasing that a fixed landscape requires less computation (Table 7).

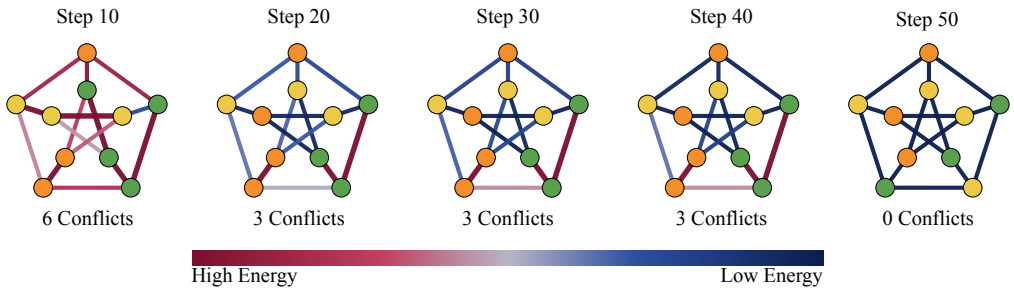

Figure 3: **Graph coloring refinement.** Conflicting edges decrease from six to zero in this example. Node colors show decoded assignments. Edge colors indicate learned local energies, from high (red) to low (blue).

Qualitative Results. Figure 3 illustrates how the composed energy guides a candidate toward a valid coloring. Local energies change between predictions with identical decoded node colors. Search can therefore continue to refine the continuous assignment while the discrete conflict count remains unchanged. As energy continues to decrease, conflicts resolve after decoding.

Ablation Study. Contrastive supervision reduces conflicts, and all seven evaluated search methods outperform DiffEBM on average. Increasing the candidate count improves results, with diminishing gains beyond 128 candidates. Even eight candidates outperform DiffEBM using 128. Full ablations appear and additional results in Appendix B.1.

4.1.2 N-Queens

Task and Setup. N-Queens tests whether composed energies for rows, columns, and diagonals can jointly enforce constraints. The task requires placing N queens on an $N \times N$ board so that no two queens attack each other. Following Oarga and Du [42], EMS learns energies from valid row configurations and applies them across the board. We train separate models for 8-, 9-, and 10-Queens because the local input representation depends on board size. For 8-Queens, we compare against LwD [1], GFlowNets [61], DIFUSCO [52], Fast T2T [36], and DiffEBM, using baseline results from Oarga and Du [42]. The main EMS comparison uses GD with 128 candidates and 20 refinement steps, compared to the 1,024 candidates of DiffEBM.

Quantitative Results. EMS solves all 100 trials of 8-Queens, compared with 97 for DiffEBM, while using 128 candidates rather than 1,024 (Table 2). Its total evaluation count is 2,688 per instance, compared with 5,377,024 for DiffEBM with PEM, approximately 2,000$\times$ fewer evaluations (Table 7). In a separate refinement-step sweep, larger boards require more steps to reach full success under shared inference settings (Figure 4). Additional search improves success within each board size as the input dimensionality increases, without retraining.

Qualitative Results. Figure 5 shows an example in which refinement resolves conflicts across the board and reaches a valid solution at step 10. Although the model is trained on local configurations, the composed objective guides changes to the complete assignment.

Model	Correct ↑	Size ↑
LwD	22	7.10
GFlowNets	14	6.93
DIFUSCO	17	6.94
Fast T2T	41	7.38
DiffEBM	97	7.97
EMS (Ours)	**100**	**8.00**

Table 2: **8-Queens results.** Correct: valid boards out of 100 trials. Size: mean number of non-conflicting queens per board (maximum 8). Baseline results are from Oarga and Du [42].

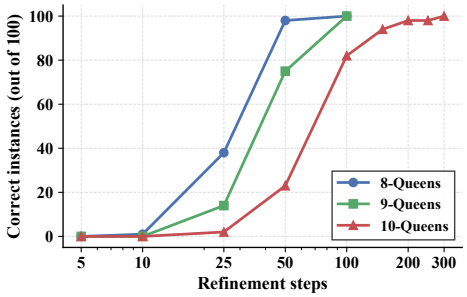

Figure 4: **N-Queens refinement budget.** Valid boards out of 100 trials versus refinement steps. All board sizes use the inference settings chosen for 10-Queens.

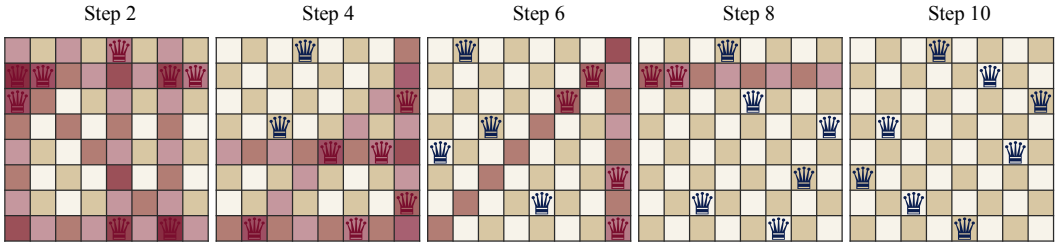

Figure 5: **8-Queens refinement.** Intermediate boards show the eight queen predicted cells before conflict filtering, with the number of attacking queen pairs below each board.

Ablation Study. As in graph coloring, contrastive supervision helps the learned energy support solution search: removing it reduces success from 100 to 77 out of 100 trials. Searching more candidates also helps. Increasing the candidate count from 64 to 128 raises success from 89 to 100. Full configurations and additional ablations appear in Appendix B.2.

4.1.3 3-SAT

Task and Setup. 3-SAT requires assigning Boolean values so that every clause, each containing three literals, is satisfied. Following Oarga and Du [42], EMS learns clause-level energies and sums them to search over complete assignments. We evaluate 100 SATLIB formulas with 20 variables and 91 clauses [27]. Baselines include GCN [31], DGL, DIFUSCO [52], Fast T2T [36], NeuroSAT [47], NSNet [37], and DiffEBM, using results reported by Oarga and Du [42]. Our main EMS and DiffEBM comparison uses clause-level pretraining without full-formula fine-tuning. EMS uses GD with 1,024 candidates and 100 refinement steps across three seeds.

Quantitative Results. EMS solves an average of 64.7 formulas, compared with 57 for pretrained DiffEBM (Table 3). DiffEBM achieves a slightly higher satisfied-clause percentage, at 99.51% compared with 99.41% for EMS. Thus, EMS finds more complete solutions, although it does not improve both metrics. With both methods using 1,024 candidates, EMS requires 103,424 total evaluations per formula, compared with 5,377,024 for DiffEBM with PEM, approximately a 52× reduction (Table 7). DiffEBM reports solving 91 of 100 formulas after additional full-formula fine-tuning [42]. That procedure is absent from the released code repository, so our comparison is restricted and evaluated to its pretraining-only setting.

Ablation Study. Contrastive supervision is essential in the evaluated 3-SAT setting: without it, EMS solves no formulas, consistent with its importance in Oarga and Du [42]. Increasing the candidate count improves success, with little further gain beyond 512 candidates (Figure 6). Full ablations appear in Appendix B.3.

4.1.4 Crossword Puzzles

Task and Setup. Crosswords extend the preceding constraint-satisfaction tasks to predictions conditioned on language clues. Each word must answer its clue and agree with intersecting words

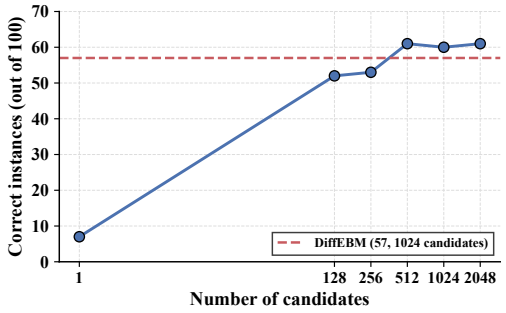

Figure 6: **3-SAT candidate-count ablation.** Single-run success versus candidate count. The dashed line shows pretrained DiffEBM's reported result with 1,024 candidates.

Model	Correct ↑	Clauses (%) ↑
GCN	5	96.17
DGL	10	95.20
DIFUSCO	6	97.34
Fast T2T	22	97.60
NeuroSAT	8	97.42
NSNet	58	98.56
DiffEBM	57	**99.51**
EMS (Ours)	$\mathbf{64.7 \pm 0.3}$	99.41 ± 0.03

Table 3: **3-SAT results.** Correct: solved formulas out of 100. Clauses: percentage of satisfied clauses. EMS reports mean ± standard error over three seeds. DiffEBM is evaluated without full-formula fine-tuning.

at shared letters. We evaluate on the Mini Crosswords benchmark [59]. Following Oarga and Du [42], EMS learns a word-level energy conditioned on a precomputed clue embedding and composes these energies over the rows and columns. We compare against input-output prompting (IO), Chain of Thought (CoT), Tree of Thoughts (ToT) [59], and DiffEBM [42], using their reported results. EMS uses GD with one candidate and 500 refinement steps. Dataset and training details appear in Appendix B.4.

Model	Letter (%) ↑	Word (%) ↑
IO	38.70	14.00
CoT	40.60	15.60
ToT	78.00	**60.00**
DiffEBM	**80.40**	50.50
EMS (Ours)	**80.40**	51.00

Table 4: **Crossword results.** Accuracy at the letter and whole-word levels. EMS uses one candidate, compared with 1,024 candidates for DiffEBM. EMS matches DiffEBM's letter accuracy, while ToT achieves the highest word accuracy.

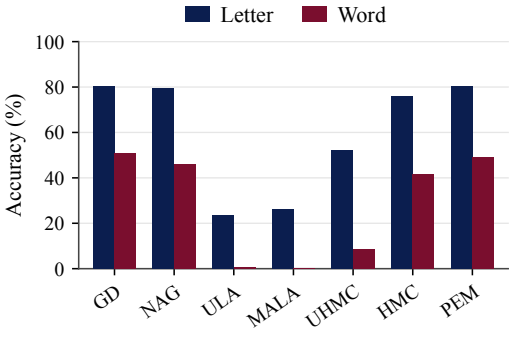

Figure 7: **Crossword search methods.** Letter and word accuracy using the same trained energy with one candidate and different search methods.

Quantitative Results. EMS matches DiffEBM's letter accuracy and achieves slightly higher word accuracy (Table 4). It obtains these results with a single candidate: 501 total evaluations per puzzle, compared with 5,377,024 for DiffEBM with PEM, approximately four orders of magnitude fewer (Table 7). ToT achieves higher word accuracy at 60.0%, despite its lower letter accuracy of 78.0%.

Qualitative Results. Figure 8 shows refinement from a random initialization to a grid with 23 of 25 letters and 7 of 10 words correct. The first update recovers several recognizable entries, and later updates correct additional letters. The two remaining letter errors affect three words because entries overlap, illustrating the difference between letter and word accuracy.

Ablation Study. Search method matters more than candidate count in this task. GD and PEM perform similarly, while ULA and MALA achieve at most 0.5% word accuracy under the evaluated settings (Figure 7). Unlike N-Queens and 3-SAT, increasing the candidate count from one to 1,024 provides little benefit. Full configurations and ablations appear in Appendix B.4.

4.2 Planning

4.2.1 Shortest Path Finding

Task and Setup. Following Du et al. [17], we evaluate shortest-path planning in directed graphs, training on 15-node graphs and testing on held-out 15-node and larger 25-node graphs. Given a graph,

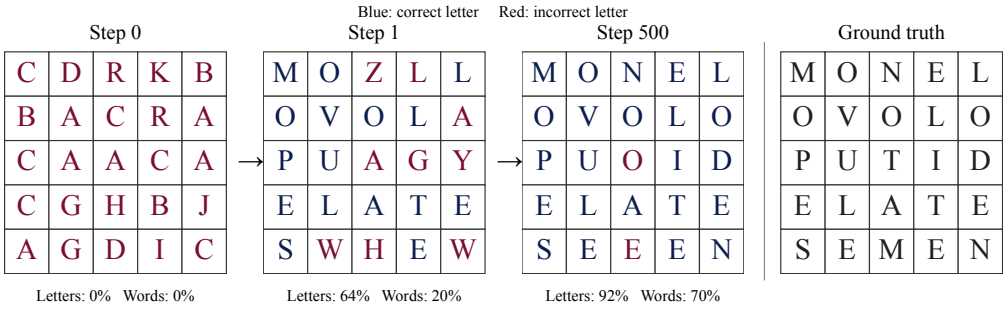

Figure 8: **Crossword refinement.** A selected example shows decoded predictions during energy minimization. Blue letters match the ground truth while red letters do not. Two letter errors remain in the final prediction.

start, and goal, the model predicts a sequence of node visits. We score only the first action, which must follow a valid edge and decrease the shortest-path distance to the goal. We compare with the diffusion, IREM, and IRED results reported by Du et al. [17]. EMS uses the same spatial-temporal graph convolution backbone without time conditioning and refines eight candidates with ULA followed by GD (ULA+GD).

Model	Similar ↑	Larger ↑
Diffusion	45.2	46.9
IREM	90.4	88.4
IRED	92.6	91.9
EMS (Ours)	**96.3**	**93.1**

Table 5: **Shortest-path results.** First-action accuracy (%) on 15-node (Similar) and 25-node (Larger) graphs. All models are trained on 15-node graphs. Baseline results are from Du et al. [17].

Method	Similar ↑	Larger ↑
GD	93.1	92.5
NAG	87.1	69.8
ULA	91.4	86.9
MALA	90.9	85.2
UHMC	90.2	86.2
HMC	90.3	85.9
ULA+GD	**96.3**	**93.1**

Table 6: **Shortest-path search methods.** First-action accuracy (%) using different search methods on the trained EMS energy. All methods perform well relative to baselines.

Quantitative Results. EMS achieves 96.3% first-action accuracy on 15-node graphs and 93.1% on 25-node graphs, slightly improving on IRED by 3.7 and 1.2 percentage points, respectively (Table 5). These gains use a larger search budget: EMS refines eight candidates with 1,848 total model evaluations, compared with one candidate and 110 evaluations for the reported IRED configuration (Table 7). The same trained EMS model is used for both graph sizes, showing that the learned energy can guide action selection on graphs larger than those seen during training. This differs from N-Queens, where the input representation requires separate models for each board size.

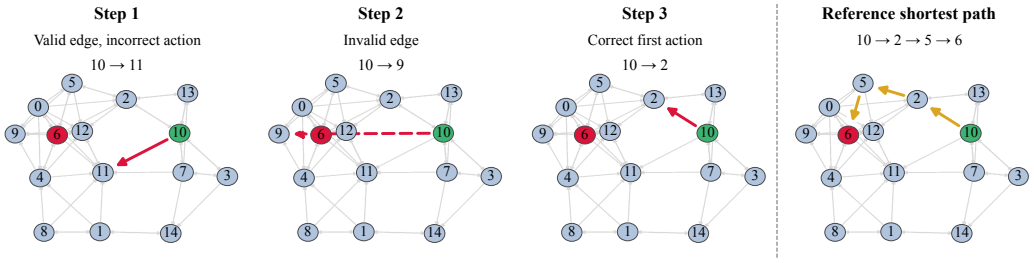

Figure 9: **Shortest-path refinement.** The green node denotes the start and the red node denotes the goal. Arrows illustrate the predicted first action during refinement, with a reference shortest path shown on the right.

Qualitative Results. Figure 9 illustrates how the decoded first action changes during refinement. An intermediate prediction may follow a valid edge without moving closer to the goal, or select an edge that does not exist. The final prediction selects a valid action along a shortest path, satisfying both parts of the evaluation criterion.

Ablation Study. ULA followed by GD achieves the highest accuracy among the evaluated search methods, while GD alone remains competitive on larger graphs (Table 6). Increasing the candidate count from one to eight improves accuracy on both graph sizes. Contrastive supervision has mixed

effects: it improves accuracy on larger graphs but reduces it on 15-node graphs. Full configurations and ablations appear in Appendix C.1.

4.2.2 Maze Solving

Task and Setup. We evaluate continuous trajectory planning on the Ghugare PointMaze suite [19]. Given start and goal configurations, the planner generates a fixed-horizon waypoint trajectory executed by a low-level controller. We compare against Decision Diffuser (DD) [2], a diffusion planner rather than an explicit EBM, and report goal-reaching success over five seeds on Medium and Large mazes. Full settings appear in Appendix C.2.

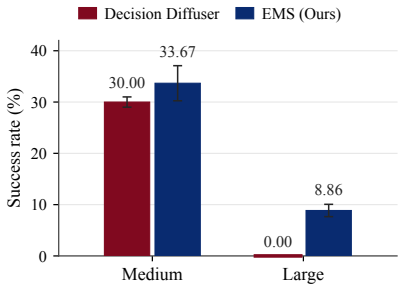

Figure 10: **PointMaze planning.** Mean goal-reaching success across five seeds. Error bars indicate standard deviations.

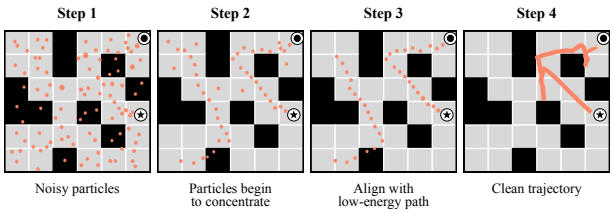

Figure 11: **Maze refinement schematic.** Illustration of waypoint refinement from noise toward a start-to-goal trajectory. The start is marked by a circle and the goal by a star.

Quantitative Results. EMS achieves 33.67% mean success on Medium mazes and 8.86% on Large mazes, compared with 30.00% and 0.00% for DD (Figure 10). These results extend the energy-minimization formulation to continuous trajectory planning, but success remains low on the larger environment. Producing reliable long-horizon plans is therefore still a challenge for EMS.

Ablation Study. ULA performs best on Medium mazes, while GD performs best on Large mazes among the evaluated search methods. The preferred EqM target scale and inference step size also vary by environment. Full ablations appear in Appendix C.2.

Compositional Planning and Limitations. We evaluate whether EMS can join short trajectory segments into longer plans using the CompDiffuser framework [39] on OGBench PointMaze `stitch`. Both methods achieve 100% success on Medium, but EMS reaches 50% on Large and 0% on Giant, compared with 100% and 68% for CompDiffuser. During training, neighboring segments come from the same trajectory, so their overlapping portions already match. At inference, the generated segments may not match, making them difficult to join into a valid path. We add noise to neighboring segments during training to improve generalization. This improves Large-maze success to 70%, but Giant remains unsolved. On larger mazes, generated plans still take shortcuts through walls rather than following corridors, leading to failed attempts. Full results appear in Appendix C.3.

5 Limitations and Conclusion

Limitations. Our current approach has several limitations. First, inference starts from Gaussian noise and can require many refinement steps or candidates. It would be interesting to explore learned initializations or guided optimizers that reduce this computation. Second, composing local energies does not always produce consistent solutions over long planning horizons. An interesting direction for future work is to fine-tune the composed energy on complete trajectories, allowing training to account for interactions between neighboring segments.

Conclusion. In this work, we present EMS, a unified framework for reasoning and planning through energy minimization. We show that a shared training objective supports both reasoning and planning.. Future work could explore fine-tuning composed energies to solve complex problems and to make plans assembled from short trajectory segments more reliable over longer horizons.

Acknowledgments and Disclosure of Funding

This work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence at Harvard University. This work was also supported by a research grant from Lambda, Inc., which provided computational resources.

References

[1] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets, 2020. URL https://arxiv.org/abs/2006.09607. 6

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making?, 2023. URL https://arxiv.org/abs/2211.15657. 2, 5, 10, 18, 27

[3] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks, 2021. URL https://arxiv.org/abs/1703.00443. 2

[4] Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning, 2026. URL https://arxiv.org/abs/2605.19376. 2

[5] Michal Balcerak, Tamaz Amiranashvili, Antonio Terpin, Suprosanna Shit, Lea Bogensperger, Sebastian Kaltenbach, Petros Koumoutsakos, and Bjoern Menze. Energy matching: Unifying flow matching and energy-based models for generative modeling, 2025. URL https://arxiv.org/abs/2504.10612. 2

[6] Michal Balcerak, Suprosanna Shit, Chinmay Prabhakar, Sebastian Kaltenbach, Michael S. Albergo, Yilun Du, and Bjoern Menze. Graph energy matching: Transport-aligned energy-based modeling for graph generation, 2026. URL https://arxiv.org/abs/2603.23398. 2

[7] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407. 2

[8] Leon Bottou. From machine learning to machine reasoning, 2011. URL https://arxiv.org/abs/1102.1808. 1

[9] Joao Carvalho, An T. Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models, 2024. URL https://arxiv.org/abs/2308.01557. 2

[10] Chang Chen, Fei Deng, Kenji Kawaguchi, Caglar Gulcehre, and Sungjin Ahn. Simple hierarchical planning with diffusion, 2024. URL https://arxiv.org/abs/2401.02644. 2

[11] Hongyi Chen, Yilun Du, Yiye Chen, Joshua Tenenbaum, and Patricio A. Vela. Planning with sequence models through iterative energy minimization, 2023. URL https://arxiv.org/abs/2303.16189. 1, 2

[12] Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolić. Xlvin: executed latent value iteration nets, 2020. URL https://arxiv.org/abs/2010.13146. 5

[13] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003. 1

[14] Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models, 2020. URL https://arxiv.org/abs/1903.08689. 1

[15] Yilun Du, Toru Lin, and Igor Mordatch. Model based planning with energy based models, 2021. URL https://arxiv.org/abs/1909.06878. 1, 2

[16] Yilun Du, Shuang Li, Joshua B. Tenenbaum, and Igor Mordatch. Learning iterative reasoning through energy minimization, 2022. URL https://arxiv.org/abs/2206.15448. 1, 2

[17] Yilun Du, Jiayuan Mao, and Joshua B. Tenenbaum. Learning iterative reasoning through energy diffusion, 2024. URL https://arxiv.org/abs/2406.11179. 1, 2, 4, 5, 8, 9, 17, 25, 26

[18] Xiaolin Fang, Caelan Reed Garrett, Clemens Eppner, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Dimsam: Diffusion models as samplers for task and motion planning under partial observability, 2024. URL https://arxiv.org/abs/2306.13196. 2

[19] Raj Ghugare, Matthieu Geist, Glen Berseth, and Benjamin Eysenbach. Closing the gap between td learning and supervised learning – a generalisation point of view, 2024. URL https://arxiv.org/abs/2401.11237. 10, 27

[20] Nikolaos Gkanatsios, Ayush Jain, Zhou Xian, Yunchu Zhang, Christopher Atkeson, and Katerina Fragkiadaki. Energy-based models are zero-shot planners for compositional scene rearrangement, 2024. URL https://arxiv.org/abs/2304.14391. 1, 2

[21] Alexi Gladstone, Ganesh Nanduru, Md Mofijul Islam, Peixuan Han, Hyeonjeong Ha, Aman Chadha, Yilun Du, Heng Ji, Jundong Li, and Tariq Iqbal. Energy-based transformers are scalable learners and thinkers, 2025. URL https://arxiv.org/abs/2507.02092. 2

[22] Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983. 2

[23] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401. 2

[24] Alec Helbling, Andrey Bryutkin, Mauro Martino, Duen Horng Chau, Nima Dehmamy, and Hendrik Strobelt. Flow reasoning models: Turning flows into efficient recurrent reasoners, 2026. URL https://arxiv.org/abs/2606.29150. 2

[25] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002. doi: 10.1162/089976602760128018. 1, 2

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239. 2

[27] Holger H. Hoos, Thomas Stützle, Ian P. Gent, Hans van Maaren, and Toby Walsh. Satlib: An online resource for research on sat, 2000. URL https://www.cs.ubc.ca/~hoos/Publ/sat2000-satlib.pdf. 7, 21

[28] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. URL http://jmlr.org/papers/v6/hyvarinen05a.html. 2

[29] Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis, 2022. URL https://arxiv.org/abs/2205.09991. 2

[30] Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871. 2

[31] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907. 5, 7

[32] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. 1

[33] Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu-Jie Huang. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006. 1, 3

[34] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems, 2019. URL https://arxiv.org/abs/1903.04598. 5

[35] Wenhao Li, Xiangfeng Wang, Bo Jin, and Hongyuan Zha. Hierarchical diffusion for offline decision making. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 20035–20064. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/li23ad.html. 2

[36] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization, 2025. URL https://arxiv.org/abs/2502.02941. 6, 7

[37] Zhaoyu Li and Xujie Si. Nsnet: A general neural probabilistic framework for satisfiability problems, 2022. URL https://arxiv.org/abs/2211.03880. 7

[38] Yunhao Luo, Chen Sun, Joshua B. Tenenbaum, and Yilun Du. Potential based diffusion motion planning, 2024. URL https://arxiv.org/abs/2407.06169. 2

[39] Yunhao Luo, Utkarsh A. Mishra, Yilun Du, and Danfei Xu. Generative trajectory stitching through diffusion composition, 2025. URL https://arxiv.org/abs/2503.05153. 2, 5, 10, 28, 29

[40] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming, 2018. URL https://arxiv.org/abs/1805.10872. 2

[41] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving, 2020. URL https://arxiv.org/abs/2007.06477. 2

[42] Alexandru Oarga and Yilun Du. Generalizable reasoning through compositional energy minimization, 2025. URL https://arxiv.org/abs/2510.20607. 1, 2, 4, 5, 6, 7, 8, 16, 17, 18, 20, 21, 22, 23, 24

[43] Scott Reed and Nando de Freitas. Neural programmer-interpreters, 2016. URL https://arxiv.org/abs/1511.06279. 2

[44] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving, 2017. URL https://arxiv.org/abs/1705.11040. 2

[45] Meir Roketlishvili, Semyon Semenov, Maksim Bobrin, Viktor Kovalchuk, Albert Baichorov, Abduragim Shtanchaev, Fakhri Karray, Dmitry V. Dylov, Martin Takáč, and Arip Asadulaev. Convex compositional reasoning models, 2026. URL https://arxiv.org/abs/2605.23395. 2

[46] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2021. 1

[47] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a sat solver from single-bit supervision, 2019. URL https://arxiv.org/abs/1802.03685. 7

[48] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585. 2

[49] Yang Song and Diederik P. Kingma. How to train your energy-based models, 2021. URL https://arxiv.org/abs/2101.03288. 2

[50] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation, 2019. URL https://arxiv.org/abs/1905.07088. 2

[51] Tao Sun, Utkarsh Aashu Mishra, Jiaxin Lu, Danfei Xu, and Iro Armeni. Energy-based compositional diffusion planning, 2026. URL https://arxiv.org/abs/2606.21646. 2

[52] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization, 2023. URL https://arxiv.org/abs/2302.08224. 6, 7

[53] Mokanarangan Thayaparan, Marco Valentino, and André Freitas. A differentiable integer linear programming solver for explanation-based natural language inference, 2024. URL https://arxiv.org/abs/2404.02625. 2

[54] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903. 5

[55] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2026. URL https://arxiv.org/abs/2506.21734. 2

[56] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver, 2019. URL https://arxiv.org/abs/1905.12149. 2

[57] Runqian Wang and Yilun Du. Equilibrium matching: Generative modeling with implicit energy-based models, 2025. URL https://arxiv.org/abs/2510.02300. 2, 3

[58] Cheng-Fu Yang, Haoyang Xu, Te-Lin Wu, Xiaofeng Gao, Kai-Wei Chang, and Feng Gao. Planning as in-painting: A diffusion-based embodied task planning framework for environments under uncertainty, 2023. URL https://arxiv.org/abs/2312.01097. 2

[59] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601. 8, 23, 24

[60] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning, 2025. URL https://arxiv.org/abs/2410.14157. 2

[61] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. In *Advances in Neural Information Processing Systems 36*, NeurIPS 2023, page 11952–11969. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/075280-0522. URL http://dx.doi.org/10.52202/075280-0522. 6

[62] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms, 2016. URL https://arxiv.org/abs/1511.08228. 2

A Additional Method Details

A.1 Energy Parameterization and Training

Energy Parameterization. We parameterize the scalar energy using the squared norm of a neural network output:

$$E_\theta(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \left\| \mathbf{z}_\theta(\mathbf{x}, \mathbf{y}) \right\|_2^2, \tag{9}$$

where $\mathbf{z}_\theta$ may be scalar- or vector-valued, depending on the backbone. This construction yields a nonnegative energy, whose gradient with respect to the candidate is

$$\nabla_{\mathbf{y}} E_\theta(\mathbf{x}, \mathbf{y}) = J_{\mathbf{z}_\theta, \mathbf{y}}^\top \mathbf{z}_\theta(\mathbf{x}, \mathbf{y}), \tag{10}$$

where $J_{\mathbf{z}_\theta, \mathbf{y}}$ is the Jacobian of the network output with respect to $\mathbf{y}$. We compute this gradient through automatic differentiation without explicitly constructing the Jacobian. The conditions $\mathbf{x}$ remain fixed during candidate refinement.

Backbone Architectures. We retain the corresponding diffusion baselines' backbone architectures while removing diffusion-time conditioning. The reasoning models use residual MLPs. Each residual block contains LayerNorm without affine parameters, ReLU activations, and three linear layers. Hidden widths, output dimensions, and conditioning inputs depend on the task. For planning, we use the spatial-temporal graph convolutional backbone of IRED for shortest paths and the trajectory U-Net backbone of Decision Diffuser for PointMaze. Models are trained separately for each task. The shared components are the energy formulation and training objective, rather than a common set of learned parameters. Architecture details appear in Appendices B and C.

Gradient Supervision. For each training example, we independently sample Gaussian noise and an interpolation coefficient $\gamma \sim \mathcal{U}(0, 1)$, then construct the corrupted candidate using Eq. (2). The EqM target uses the fixed decay function $c(\gamma) = 1 - \gamma$. Neither γ nor a diffusion timestep is provided to the energy network.

Training first differentiates the scalar energy with respect to the corrupted candidate, retaining the computation graph. Backpropagation through the EqM loss then differentiates this gradient with respect to the model parameters. This requires mixed derivatives of the energy, but does not unroll a candidate refinement trajectory. The reasoning implementation uses mean squared error averaged over the batch and candidate coordinates.

Contrastive Supervision. When contrastive supervision is enabled, the EqM and contrastive terms share the same corrupted positive example. Each associated negative uses the positive's interpolation coefficient and Gaussian noise, as described in Section 3.2. We compute the contrastive term using cross-entropy over negative energy scores, with the positive designated as the correct class. Negatives are constructed directly from task-specific perturbations rather than generated by an inner search chain. The objective is $\mathcal{L} = \mathcal{L}_{\mathrm{EqM}} + \beta \mathcal{L}_{\mathrm{CL}}$. Crossword training uses $\beta = 0$. Perturbation rules and contrastive weights are reported with the corresponding task configurations.

Optimization and Checkpoints. The reasoning models are optimized with AdamW, and we maintain an exponential moving average (EMA) of the model parameters. The evaluated checkpoint may use the ordinary or EMA weights, as specified for each task. In the released reasoning training scripts, graph coloring and N-Queens checkpoints are selected by the lowest training loss, while 3-SAT and crossword checkpoints are selected by the lowest validation loss. Learning rates, weight decay, batch sizes, training duration, EMA settings, and evaluation configurations appear in Appendices B and C.

A.2 Search Methods

Section 3 describes GD, ULA, and PEM. Here we provide additional search method details. For a fixed instance $\mathbf{x}$, let $E(\mathbf{y})$ denote the full-instance or composed energy, and omit the candidate index when updates operate independently. All methods refine candidates on the same fixed objective. Their effectiveness depends on the task and search settings.

Nesterov Accelerated Gradient (NAG). NAG uses momentum and evaluates the gradient at a lookahead position:

$$\begin{aligned} \mathbf{v}^{k+1} &= \mu \mathbf{v}^k - \eta \nabla E(\mathbf{y}^k + \mu \mathbf{v}^k), \\ \mathbf{y}^{k+1} &= \mathbf{y}^k + \mathbf{v}^{k+1}, \end{aligned} \tag{11}$$

where $\mathbf{v}^0 = \mathbf{0}$, $\eta > 0$ is the step size, and $\mu \in [0, 1)$ is the momentum coefficient.

Metropolis-Adjusted Langevin Algorithm (MALA). MALA augments a Langevin proposal with an acceptance test. Given a candidate $\mathbf{y}$, we draw $\mathbf{y}'$ from

$$q(\mathbf{y}' \mid \mathbf{y}) = \mathcal{N}(\mathbf{y}'; \mathbf{y} - \eta \nabla E(\mathbf{y}), 2\eta \mathbf{I}), \tag{12}$$

where $\eta > 0$ is the step size. The proposal is accepted with probability

$$\alpha(\mathbf{y}, \mathbf{y}') = \min\left\{1, \frac{\exp[-E(\mathbf{y}')]q(\mathbf{y} \mid \mathbf{y}')}{\exp[-E(\mathbf{y})]q(\mathbf{y}' \mid \mathbf{y})}\right\}. \tag{13}$$

Otherwise, the candidate remains unchanged. The proposal correction requires evaluating both the energy and its gradient at the proposed candidate, in addition to those at the current candidate.

Hamiltonian Methods (UHMC and HMC). These methods introduce an auxiliary momentum $\mathbf{r}$ and define

$$H(\mathbf{y}, \mathbf{r}) = E(\mathbf{y}) + \frac{1}{2}\|\mathbf{r}\|_2^2. \tag{14}$$

Momentum is sampled from a standard Gaussian or partially refreshed according to

$$\mathbf{r} \leftarrow \rho \mathbf{r} + \sqrt{1 - \rho^2}\, \boldsymbol{\xi}, \qquad \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{15}$$

where $\rho \in [0, 1)$ controls momentum persistence. Starting from the refreshed momentum, each proposal performs L leapfrog steps:

$$\begin{aligned} \mathbf{r}^{\ell+\frac{1}{2}} &= \mathbf{r}^{\ell} - \frac{\eta}{2}\nabla E(\mathbf{y}^{\ell}), \\ \mathbf{y}^{\ell+1} &= \mathbf{y}^{\ell} + \eta \mathbf{r}^{\ell+\frac{1}{2}}, \\ \mathbf{r}^{\ell+1} &= \mathbf{r}^{\ell+\frac{1}{2}} - \frac{\eta}{2}\nabla E(\mathbf{y}^{\ell+1}). \end{aligned} \tag{16}$$

UHMC retains the resulting position without an acceptance test. HMC accepts the proposed position with probability

$$\alpha = \min\left\{1, \exp\left[H(\mathbf{y}^0, \mathbf{r}^0) - H(\mathbf{y}^L, \mathbf{r}^L)\right]\right\}, \tag{17}$$

and otherwise retains the previous position. The step size, leapfrog count, and momentum settings are specified with the corresponding task configurations.

PEM Implementation. PEM resamples candidates with replacement using weights proportional to $\exp[-E(\mathbf{y}_p)]$, as described in the main text [42]. Our implementation computes energies and gradients before resampling and reuses each selected ancestor's gradient for its descendants. Independent Gaussian perturbations then diversify duplicated candidates. The perturbation scale is either specified directly as σ or set to $\sqrt{2\eta}$ for the ULA-based variant.

Langevin Refinement Followed by GD (ULA+GD). This procedure applies ULA for K_{ULA} updates, then continues from the resulting candidates with K_{GD} gradient descent updates. The energy remains fixed across both phases. Langevin updates provide initial stochastic exploration, while GD refines the resulting candidates without additional noise.

Evaluation and Interpretation. We use these methods as finite-budget search procedures, selecting and decoding the final candidate with the lowest energy rather than returning calibrated Gibbs samples. Task-specific configurations and comparisons appear in Appendices B and C. Because acceptance tests and leapfrog integration can require multiple model evaluations per update, equal update counts need not imply equal inference budgets. Appendix A.3 details our accounting.

A.3 Inference Budget

We report model forward evaluations (NFE) per candidate, candidate counts, and their product for the configurations used in the main comparisons. NFE includes evaluations used for refinement, resampling, and final energy-based selection. For compositional models, one evaluation covers all factors of the objective for one candidate. The candidate weighted total is

$$N_{\text{total}} = P\, N_{\text{candidate}}, \tag{18}$$

Task	Method	NFE per candidate	Candidates	Total evaluations
Graph coloring	DiffEBM + PEM	502,501	128	64,320,128
	EMS	501	128	64,128
8-Queens	DiffEBM + PEM	5,251	1,024	5,377,024
	EMS	21	128	2,688
3-SAT	DiffEBM + PEM	5,251	1,024	5,377,024
	EMS	101	1,024	103,424
Crosswords	DiffEBM + PEM	5,251	1,024	5,377,024
	EMS	501	1	501
Path finding	IRED	110	1	110
	EMS	231	8	1,848

Table 7: **Inference evaluation budgets.** NFE includes refinement and energy-scoring evaluations per candidate. Total evaluations multiply this count by the candidate count. DiffEBM counts correspond to the released PEM configurations, and IRED uses its released default inference configuration. EMS counts correspond to the main reported settings.

where P is the candidate count. Candidates and local factors can be evaluated in batches, so this total does not count sequential batched calls. Planning budgets refer to one plan generation call and exclude controller evaluations and repeated replanning over an episode.

DiffEBM with PEM. The released PEM implementation [42] performs a denoising rollout at each diffusion timestep, followed by an additional denoising update. With T diffusion timesteps, this requires

$$N_{\text{denoise}} = \sum_{i=1}^{T} i + T = \frac{T(T+1)}{2} + T \tag{19}$$

denoising evaluations per candidate. It also evaluates energies once per timestep for resampling and once after refinement for final selection. Thus, for the configurations reported here,

$$N_{\text{DiffEBM+PEM}} = \frac{T(T+1)}{2} + 2T + 1. \tag{20}$$

Graph coloring uses $T = 1{,}000$, giving 501,500 denoising evaluations, 1,000 resampling evaluations, and one final scoring evaluation, for 502,501 evaluations per candidate. N-Queens, 3-SAT, and crosswords use $T = 100$, giving 5,150 denoising evaluations, 100 resampling evaluations, and one final scoring evaluation, for 5,251 in total. These counts describe the released PEM procedure rather than diffusion inference in general.

EMS. The main reasoning configurations use GD or ULA, each requiring one energy evaluation per refinement update. An additional evaluation scores the final candidates for selection, giving $N_{\text{EMS}} = K + 1$ for K updates. Graph coloring and crosswords use 500 updates, 8-Queens uses 20, and 3-SAT uses 100. The path-finding configuration uses 200 ULA updates followed by 30 GD updates and one final scoring evaluation, giving 231 evaluations per candidate.

These counts apply to the stated search configurations. Other methods can require more evaluations per update: MALA evaluates both the current and proposed candidates, and Hamiltonian methods evaluate gradients across multiple leapfrog steps. Equal refinement-step counts therefore need not imply equal NFE.

IRED. The released default IRED configuration [17] uses 10 diffusion timesteps. Each timestep performs one reverse denoising evaluation followed by five refinement updates. Each refinement update requires one evaluation for the energy gradient and another to score the proposed candidate. The resulting count is

$$N_{\text{IRED}} = 10(1 + 2 \cdot 5) = 110. \tag{21}$$

The reported EMS path-finding configuration therefore uses more model evaluations per candidate and more candidates than the released default IRED configuration.

PointMaze Planning. Table 8 reports the plan generation step budgets for the full-instance PointMaze experiments. We report these separately because converting steps into forward evaluations depends on guidance and final-scoring settings.

Environment	Method	Steps	Candidates
PointMaze Medium	Decision Diffuser	512	1
	EMS	50	1
PointMaze Large	Decision Diffuser	512	1
	EMS	75	1

Table 8: **PointMaze plan generation budgets.** Steps denote denoising timesteps for Decision Diffuser and refinement updates for EMS. These are step counts, rather than verified total forward-evaluation counts.

These step budgets do not establish exact forward-evaluation counts. Decision Diffuser's released implementation [2] uses one denoiser evaluation per timestep without guidance and two with guidance. For EMS, final energy scoring can add an evaluation beyond the refinement updates. Because these evaluator settings have not been verified for the reported PointMaze runs, we report step counts only and exclude PointMaze from Table 7.

B Reasoning Experiments

B.1 Graph Coloring

Data and Representation. We follow the graph-coloring benchmark of Oarga and Du [42], evaluating five graphs from each of eight distributions: small and large Erdős–Rényi, Holme–Kim, and regular expander graphs, together with Paley and complete graphs. The supplied chromatic number determines the number of available colors at evaluation.

Training uses local colored edges rather than complete graphs. Each endpoint is represented by a 14-dimensional one-hot vector, giving a 28-dimensional edge representation. The training set contains all $14 \times 13 = 182$ ordered pairs of distinct colors. We sample these pairs with replacement, drawing 50,000 examples per training epoch. Consequently, training does not require solutions for the complete evaluation graphs.

Model and Training. The energy network is a residual MLP with embedding width 256, three residual blocks, MLP ratio 2, and scalar output before the squared-energy transformation described in Appendix A.1. We train for 1,024 epochs with batch size 1,024 using AdamW, learning rate 10^{-4}, and weight decay 0.01. The main configuration uses $\lambda = 4$, $\beta = 1$, and $c(\gamma) = 1 - \gamma$. We maintain EMA weights with decay 0.9999, save the checkpoint with the lowest training loss, and use its EMA weights for evaluation.

Contrastive Negatives. Each valid edge produces two negative examples. The first assigns both endpoints the same color, chosen uniformly from the two positive colors, with each nonzero entry independently drawn from $\mathcal{U}(0.5, 1)$. The second retains the positive edge but adds an extra nonzero entry, drawn from $\mathcal{U}(0.01, 1)$, at a randomly selected inactive coordinate. These negatives represent a same-color conflict and an invalid multi-color endpoint, respectively. The positive and both negatives share the same interpolation coefficient and Gaussian noise when constructing the training losses.

Composition and Decoding. For a graph with n nodes and chromatic number k, each candidate is a continuous tensor in $\mathbb{R}^{n \times k}$. We pad each endpoint representation with zeros to 14 coordinates and concatenate the endpoints of every edge. The search objective sums the resulting local energies over the graph's unique undirected edges.

The main configuration initializes 128 candidates from Gaussian noise and applies 500 ULA updates with step size 10^{-2}. After each update, candidate coordinates are clamped to $[-1, 1]$. We select the candidate with the lowest final composed energy and decode each node's color by taking the argmax over its k coordinates. The metric is the number of edges whose endpoints receive the same decoded color.

Baselines and Reporting. The main baseline results are reproduced from Oarga and Du [42], including GCN, GAT, XLVIN, GNN-GCP, and DiffEBM. Both EMS and DiffEBM use 128 candidates in the main graph-coloring comparison. Their evaluation budgets are detailed in Appendix A.3.

Table 1 reports the mean and standard deviation across five graphs within each distribution. The summary ablations below report the unweighted mean of the eight distribution means. These averages summarize conflict counts across the benchmark; they are not uncertainties over training seeds.

Search Methods and Contrastive Supervision. Table 9 compares seven search methods with and without contrastive supervision. Within each training configuration, all methods use the same trained energy and 128 candidates with 500 updates. NAG uses momentum coefficient $\mu = 0.3$. The Hamiltonian methods use three leapfrog steps and persistent momentum with damping 0.9. PEM uses perturbation scale $\sigma = 0.1$. Equal update counts do not imply equal function-evaluation counts across these methods.

With contrastive supervision, every evaluated EMS search method achieves fewer mean conflicts than DiffEBM with PEM. ULA performs best, reducing the mean conflict count from 24.23 to 3.45. Removing contrastive supervision increases ULA's mean to 21.98. Several methods remain competitive without contrastive supervision, but combining it with gradient matching substantially improves ULA's performance.

Method	Without CL $\beta = 0$	With CL $\beta = 1$
GD	23.40	9.15
NAG	24.45	8.40
ULA	21.98	**3.45**
MALA	22.43	4.93
UHMC	20.38	4.95
HMC	**19.08**	4.78
PEM	25.45	7.78

Table 9: **Search and contrastive ablations.** Mean conflicts across eight graph distributions (lower is better). Both configurations use $\lambda = 4$, linear target decay, 128 candidates, and 500 updates. DiffEBM with PEM obtains 24.23.

Step sweep		Candidate sweep	
Updates	Conflicts	Candidates	Conflicts
100	4.95	8	4.20
200	3.98	64	3.53
300	3.78	128	3.45
400	3.50	512	**3.23**
500	**3.45**	1,024	**3.23**

Table 10: **Inference budget ablations.** Mean conflicts using ULA with $\lambda = 4$, $\beta = 1$, and linear target decay. The step sweep fixes 128 candidates. The candidate sweep fixes 500 updates. Both use step size 10^{-2} and EMA weights.

Inference Budget. Table 10 varies refinement steps and candidate count separately. At 128 candidates, increasing refinement from 100 to 500 updates reduces mean conflicts from 4.95 to 3.45. With 500 updates, eight candidates already achieve 4.20 mean conflicts, compared with 24.23 for DiffEBM using 128 candidates. Increasing EMS from 128 to 512 candidates provides a smaller improvement, from 3.45 to 3.23, with no further reduction in the aggregate mean at 1,024 candidates.

Target Scale and Contrastive Weight. We separately vary λ and β (Table 11). These sweeps use 300 refinement updates, rather than the 500 updates of the main configuration. All tested values achieve fewer mean conflicts than DiffEBM's 24.23. Within the tested target scales, $\lambda = 4$ performs best. At this scale, increasing β from 0.1 to 1 reduces mean conflicts from 4.85 to 3.78. These results demonstrate sensitivity to the training objective while showing that the advantage is not confined to a single tested setting.

Target Decay. We also compare decay functions $c(\gamma) = (1 - \gamma)^q$ while holding $\lambda = 4$ and $\beta = 1$ fixed. All tested functions vanish at the clean-solution endpoint, but their performance differs substantially. Linear decay achieves 3.45 mean conflicts, compared with 16.95–37.95 for the tested alternatives (Table 11). Thus, vanishing at clean samples alone does not determine the quality of the learned search landscape.

Target scale		Contrastive weight		Target decay	
λ	Conflicts	β	Conflicts	q	Conflicts
1	17.73	0.1	4.85	8	27.68
4	**4.53**	0.5	4.53	3	37.95
8	11.03	1	**3.78**	1	**3.45**
–	–	–	–	1/3	16.95
–	–	–	–	1/8	23.18

Table 11: **Graph coloring training objective ablations.** Mean conflicts across eight graph distributions (lower is better). All evaluations use ULA, 128 candidates, step size 10^{-2}, and EMA weights. The λ sweep fixes $\beta = 0.5$. The β sweep fixes $\lambda = 4$. Both use 300 updates and linear target decay. The decay sweep uses $c(\gamma) = (1 - \gamma)^q$, $\lambda = 4$, $\beta = 1$, and 500 updates.

EMA Weights. Under the main ULA evaluation settings, using EMA weights reduces mean conflicts from 3.78 to 3.45. We therefore use EMA weights for the main graph coloring results. This choice is task specific, as detailed in the other experiment subsections.

B.2 N-Queens

Data and Representation. Following Oarga and Du [42], EMS learns from local configurations extracted from valid N-Queens boards. Each training example is an N-dimensional binary occupancy vector containing one queen. Our data loader extracts both rows and columns from the supplied training solutions and samples 20,000 local examples with replacement per epoch.

We train separate models for 8-, 9-, and 10-Queens because the local input dimension depends on board size. These experiments evaluate composition from local structures into complete boards within each size, rather than transfer of a single trained model to larger boards.

Model and Training. Each model uses a residual MLP with embedding width 128, three residual blocks, MLP ratio 2, and scalar output before the energy transformation described in Appendix A.1. We train for 2,040 epochs with batch size 2,048 using AdamW, learning rate 10^{-4}, and weight decay 0.01. The main configuration uses $\lambda = 16$, $\beta = 0.5$, and linear target decay $c(\gamma) = 1 - \gamma$. We save and evaluate the checkpoint with the lowest training loss.

Contrastive Negatives. For each positive local configuration, we construct one negative by choosing uniformly among three perturbations: adding occupancy sampled from $\mathcal{U}(0.5, 1)$ to an unoccupied coordinate, subtracting an amount sampled from $\mathcal{U}(0.5, 1)$ from the occupied coordinate, or setting all coordinates to zero. These perturbations introduce additional occupancy or remove part or all of the existing queen's occupancy. The positive and negative share the same interpolation coefficient and Gaussian noise when constructing the losses.

Composition and Decoding. Each candidate is a continuous $N \times N$ board. We extract all rows, columns, and diagonals in both directions, padding shorter diagonals with zeros to length N. The search objective sums their learned energies, with each local term scaled by $1/4$, following the evaluation implementation.

The added padding stays fixed during search; only actual board cells are updated. There is, however, a mismatch between training and how we use the energy for diagonals. Rows and columns must contain exactly one queen, but a valid diagonal can be empty. Because training uses empty configurations as negatives, the shared energy can also penalize valid empty diagonals. Fixed padding does not remove this mismatch. We therefore use the composed energy as a guide for search, rather than an exact encoding of the rules. The decoding procedure below checks that the selected queens do not attack one another.

Candidates are initialized independently from Gaussian noise and refined without coordinate clipping. After selecting the candidate with the lowest composed energy, we rank its cells by continuous values and consider the top N cells in descending order. A queen is retained only if its placement introduces no row, column, or diagonal conflict with previously retained queens. Cells outside the top N are not considered. A decoded board is counted as correct if it contains N mutually non-attacking queens.

We report the number of correct boards out of 100 independently initialized search trials and the average number of retained queens, denoted Size. These are repeated searches for a fixed board size, rather than 100 distinct problem specifications.

Main Evaluation. The main 8-Queens comparison uses GD with 128 candidates, 20 refinement steps, and step size 0.1. This configuration solves all 100 trials. Baseline results are reproduced from Oarga and Du [42]. DiffEBM solves 97 trials using 1,024 candidates. Appendix A.3 gives the corresponding evaluation counts.

Search Methods. Table 12 compares search methods using 1,024 candidates for each board size. The 8-Queens configuration uses 20 updates with step size 0.1, 9-Queens uses 100 updates with step size 0.01, and 10-Queens uses 700 updates with step size 0.01. All use $\lambda = 16$, $\beta = 0.5$.

NAG uses momentum coefficient $\mu = 0.3$. UHMC and HMC use three leapfrog steps and persistent momentum with damping 0.9. PEM uses perturbation scale $\sigma = 0.1$. Within each board size, the

methods use the same trained energy and refinement step budget, although their numbers of model evaluations can differ.

	8-Queens		9-Queens		10-Queens	
Method	Correct	Size	Correct	Size	Correct	Size
GD	**100**	**8.00**	**100**	**9.00**	99	9.99
NAG	97	7.97	**100**	**9.00**	**100**	**10.00**
ULA	0	3.72	0	5.14	0	5.48
MALA	0	3.84	0	5.31	0	5.78
UHMC	0	4.11	0	5.32	0	5.57
HMC	4	6.20	0	5.39	0	5.86
PEM	48	7.45	30	8.13	26	9.06

Table 12: **N-Queens search method ablations.** Correct counts valid boards out of 100 trials. Size is the average number of retained non-attacking queens. All methods use 1,024 candidates. The 8-, 9-, and 10-Queens configurations use 20, 100, and 700 updates, respectively. Models are trained separately for each board size.

GD and NAG achieve the highest success rates under these settings. ULA, MALA, and UHMC produce no complete solutions, while PEM achieves intermediate success. The differences show that compatibility with a trained energy does not imply comparable search performance.

Training Objective. Table 13 varies the contrastive weight and target scale for 8-Queens. Both sweeps use GD with 128 candidates, 20 updates, and step size 0.1. At $\lambda = 16$, removing contrastive supervision reduces success from 100 to 77 trials, while increasing its weight from $\beta = 0.5$ to $\beta = 1$ reduces success to 91. With $\beta = 0.5$, target scales of 12 and 16 both achieve 100 successful trials, compared with 81 at $\lambda = 8$. Thus, contrastive supervision helps under these settings, but increasing its weight does not necessarily improve performance.

Objective sweeps		
Parameter	Value	Correct
CL weight β	0	77
	0.5	**100**
	1	91
Target scale λ	8	81
	12	**100**
	16	**100**

Table 13: **8-Queens objective ablations.** Correct boards out of 100 trials using GD, 128 candidates, 20 updates, and step size 0.1. The β sweep fixes $\lambda = 16$. The λ sweep fixes $\beta = 0.5$.

Candidate sweep		Step sweep	
Candidates	Correct	Updates	Correct
1	2	1	0
8	17	5	2
16	42	10	38
32	67	15	89
64	89	20	**100**
128	**100**	–	–

Table 14: **8-Queens inference ablations.** Correct boards out of 100 trials using GD, $\lambda = 16$, $\beta = 0.5$, and step size 0.1. The candidate sweep fixes 20 updates. The step sweep fixes 128 candidates.

Inference Budget. Table 14 varies candidate count and refinement steps separately under the main 8-Queens GD configuration. With 20 updates, increasing the candidate count from one to 128 raises success from 2 to 100 trials. With 128 candidates, increasing refinement from one to 20 updates raises success from 0 to 100 trials.

Figure 4 reports a separate sweep using the inference settings selected for 10-Queens across all three separately trained models. It therefore differs from the main 8-Queens configuration. Under those shared settings, 8- and 9-Queens reach 100 successful trials at 100 updates, while 10-Queens reaches this level at 300 updates. These results characterize the observed search requirements within each board size, rather than cross-size model generalization.

B.3 3-SAT

Data and Representation. Following Oarga and Du [42], we evaluate on 100 satisfiable formulas from the SATLIB `uf20-91` benchmark, each containing 20 variables and 91 clauses [27]. Training uses individual clauses extracted from the supplied training formulas and their satisfying assignments. Each example contains three Boolean assignment values and three binary indicators specifying which literals are negated, giving a six-dimensional network input. Training and validation use the corresponding clause-level splits.

Model and Training. The energy network is a residual MLP with embedding width 128, three residual blocks, MLP ratio 2, and scalar output before the energy transformation described in Appendix A.1. We train for 1,000 epochs with batch size 1,024 using AdamW, learning rate 10^{-4}, and no weight decay. We select the checkpoint with the lowest validation loss and evaluate its standard model weights rather than EMA weights.

The main three-seed comparison uses $\lambda = 0.007$, $\beta = 0.5$, and linear target decay $c(\gamma) = 1 - \gamma$. Unless otherwise stated, the separate single run ablations below use $\lambda = 0.005$ and $\beta = 0.5$. Their results should therefore be distinguished from the main three-seed comparison.

Contrastive Negatives. For each satisfying clause assignment, we construct the assignment that makes all three literals false. Specifically, each positive literal receives variable value zero, while each negated literal receives variable value one. The literal negation indicators remain fixed. We corrupt the positive and negative assignments using the same interpolation coefficient and Gaussian noise. EqM supervises gradients with respect to the three assignment coordinates, while contrastive supervision ranks the positive against its associated unsatisfying alternative.

Composition and Decoding. Each candidate contains one continuous value per variable. For every clause, we gather its three variable values, concatenate the literal-negation indicators, and evaluate the local energy. Summing these energies over all clauses defines the formula-level search objective.

Candidates are initialized independently from Gaussian noise. We refine them without coordinate clipping during the updates, then clamp their values to $[0, 1]$ before final energy scoring. The candidate with the lowest composed energy is selected and rounded to a Boolean assignment. We report both the number of completely solved formulas and the percentage of satisfied clauses across the test set.

Main Comparison and Seed Variability. The main EMS configuration uses GD with 1,024 candidates, 100 refinement steps, and step size 0.1. Table 15 reports results for seeds 0, 13, and 42. EMS solves 64.7 ± 0.3 formulas and satisfies $99.41 \pm 0.03\%$ of clauses, where uncertainties are standard errors across the three seeds, computed as the sample standard deviation divided by $\sqrt{3}$.

The pretrained DiffEBM baseline solves 57 formulas and satisfies 99.51% of clauses [42]. Thus, EMS solves more complete formulas in this comparison, while DiffEBM satisfies a slightly higher fraction of clauses. These results are limited to the evaluated 100-formula test set.

DiffEBM reports 91 solved formulas with additional full-formula fine-tuning. The released code does not include that fine-tuning procedure, so we do not report a matched fine-tuning comparison. Our exploratory attempts to adapt formula-level fine-tuning did not yield an improvement and do not establish equivalence to the baseline's training procedure.

Search Methods. Table 18 compares seven search methods under the separate $\lambda = 0.005$ configuration. All use the same trained energy, 1,024 candidates, 100 updates, and step size 0.1. NAG uses momentum coefficient $\mu = 0.3$. UHMC and HMC use three leapfrog steps and persistent momentum with damping 0.9. PEM uses perturbation scale $\sigma = 0.1$.

GD and NAG each solve 60 formulas under these settings. PEM solves 40, while the other evaluated methods solve fewer complete formulas despite satisfying most clauses. These results support deterministic refinement for this configuration, without implying that the same ordering holds across tasks or search budgets.

Seed	Correct	Clauses (%)
0	64	99.37
13	65	99.40
42	65	99.47
Mean $\pm$ SE	64.7 ± 0.3	99.41 ± 0.03

Table 15: **3-SAT results across seeds.** Correct counts solved formulas out of 100. Clauses denotes the percentage of satisfied clauses. All runs use $\lambda = 0.007$, $\beta = 0.5$, GD, 1,024 candidates, and 100 updates.

β	Correct
0	0
0.1	52
0.5	**60**
1	54
2	40

Table 16: **3-SAT contrastive weight ablation.** Solved formulas out of 100 using GD, $\lambda = 0.005$, 1,024 candidates, 100 updates, step size 0.1, and no weight decay.

Training Objective. Table 17 varies the EqM target scale while fixing $\beta = 0.5$. Performance is sensitive to this scale: $\lambda = 1$ solves four formulas, whereas the tested values from 0.001 to 0.05 solve

between 52 and 60. The $\lambda = 0.007$ entry belongs to this earlier single run sweep and is distinct from the three-seed result in Table 15.

Table 16 varies the contrastive weight at $\lambda = 0.005$. Removing contrastive supervision reduces success from 60 to zero solved formulas and lowers clause satisfaction from 99.38% to 89.26%. A moderate weight of $\beta = 0.5$ performs best among the tested values. Increasing the weight to 1 or 2 reduces complete solution accuracy. Under these settings, gradient matching alone is insufficient for successful formula-level search.

λ	Correct	Clauses (%)
1	4	96.71
0.5	8	97.82
0.1	45	99.21
0.05	56	99.37
0.01	57	99.29
0.007	56	99.32
0.005	**60**	**99.38**
0.003	57	99.31
0.001	52	99.18

Table 17: **3-SAT target scale ablation.** Single-run results using GD, $\beta = 0.5$, 1,024 candidates, 100 updates, step size 0.1, and no weight decay.

Method	Correct	Clauses (%)
GD	**60**	**99.38**
NAG	**60**	99.33
ULA	12	97.77
MALA	14	97.79
UHMC	0	96.96
HMC	2	97.07
PEM	40	99.00

Table 18: **3-SAT search method ablation.** Single-run results with $\lambda = 0.005$, $\beta = 0.5$, 1,024 candidates, 100 updates, and step size 0.1.

Inference Budget. Table 19 varies candidate count and refinement steps separately. With 100 updates, increasing the candidate count from one to 512 raises success from 7 to 61 formulas. Larger populations provide little further improvement in this sweep.

With 512 candidates, increasing refinement from 20 to 100 updates raises success from 34 to 61 formulas. At 120 updates, success is 60, showing that additional refinement does not guarantee a monotonic improvement in decoded solution quality. Both sweeps use the separate $\lambda = 0.005$ configuration. Appendix A.3 reports the model evaluation counts for the main comparison.

Candidate-count sweep			Refinement-step sweep		
Candidates	Correct	Clauses (%)	Updates	Correct	Clauses (%)
1	7	97.03	20	34	98.76
128	52	99.11	40	51	99.20
256	53	99.26	60	57	99.27
512	**61**	**99.41**	80	58	99.36
1,024	60	99.38	100	**61**	**99.41**
2,048	**61**	99.38	120	60	99.40

Table 19: **3-SAT inference budget ablations.** Both sweeps use GD, $\lambda = 0.005$, $\beta = 0.5$, step size 0.1, and no weight decay. The candidate-count sweep fixes 100 updates. The refinement-step sweep fixes 512 candidates. Results are from single runs.

B.4 Crossword Puzzles

Data and Representation. We evaluate on 20 puzzles from the Mini Crosswords benchmark [59], following the split and representations provided by Oarga and Du [42]. Each puzzle is a 5×5 grid containing five across and five down entries. Letter accuracy measures the fraction of the 500 grid cells predicted correctly, while word accuracy measures the fraction of the 200 complete entries predicted correctly.

Training uses individual five letter answers paired with precomputed clue embeddings. Each answer is represented by five one-hot vectors over the 26-letter alphabet, giving 130 assignment coordinates. These are concatenated with a 1,536-dimensional clue embedding to form a 1,666-dimensional network input. The clue embeddings remain fixed during training and search.

Model and Training. The energy network is a residual MLP with embedding width 1,024, three residual blocks, MLP ratio 1, and output dimension 1,666 before the energy transformation described in Appendix A.1. We train for 1,500 epochs with batch size 2,048 using AdamW, learning rate 10^{-4},

and weight decay 0.8. The main configuration uses target scale $\lambda = 8$. We select the checkpoint with the lowest validation loss and evaluate its standard model weights rather than EMA weights.

Unlike the other reasoning configurations, crossword training uses EqM without contrastive supervision, corresponding to $\beta = 0$. The objective supervises energy gradients with respect to the corrupted answer representation, conditioned on the unchanged clue embedding.

Composition and Decoding. Each candidate is a continuous tensor in $\mathbb{R}^{5\times5\times26}$. We extract its five rows and five columns, concatenate each entry with its clue embedding, and evaluate the corresponding word-level energy. The composed objective sums these ten energies with a factor of $1/2$ applied to each local term. Across and down entries share the same grid variables, so refinement jointly updates intersecting words.

The main configuration initializes one candidate from Gaussian noise and applies 500 GD updates with step size 0.1 and no step-size decay. Candidate values are not clipped during refinement. Before final scoring, values are clamped to $[-1, 1]$. When multiple candidates are used, we select the candidate with the lowest composed energy. Each cell is decoded by taking the argmax over its 26 letter coordinates.

Baselines and Reporting. We reproduce the reported IO, CoT, ToT, and DiffEBM results from prior work [59, 42]. EMS matches DiffEBM's 80.4% letter accuracy and achieves 51.0% word accuracy, compared with 50.5% for DiffEBM. ToT obtains lower letter accuracy but higher word accuracy, at 78.0% and 60.0%, respectively. EMS uses one candidate in the main comparison, whereas DiffEBM uses 1,024. Appendix A.3 reports their evaluation budgets.

The ablations below report individual evaluation runs, rather than means across training seeds. The separate training sweeps should not be interpreted as repeated measurements of the main configuration.

Search Methods. Table 20 reports the numerical results underlying the search method comparison in the main paper. All methods use the same trained energy, one candidate, 500 updates, and step size 0.1. NAG uses momentum coefficient $\mu = 0.3$. UHMC and HMC use three leapfrog steps and persistent momentum with damping 0.9. PEM uses perturbation scale $\sigma = 0.1$.

GD achieves the highest reported word accuracy, while PEM obtains similar letter accuracy. With one candidate, PEM's resampling is trivial, so this configuration evaluates its noisy gradient refinement rather than population selection. ULA and MALA perform poorly under the evaluated settings, showing that stochastic exploration does not consistently benefit crossword refinement.

Training Duration. Table 21 reports a separate training duration sweep. Word accuracy increases from 27.5% at 250 epochs to 51.0% at 1,500 epochs, with fluctuations between settings. Longer training does not consistently improve both metrics: the 2,000 epoch configuration obtains slightly higher letter accuracy but lower word accuracy than the 1,500 epoch configuration.

Method	Letter (%)	Word (%)
GD	**80.40**	**51.00**
NAG	79.40	46.00
ULA	23.40	0.50
MALA	26.00	0.00
UHMC	52.20	8.50
HMC	75.80	41.50
PEM	80.20	49.00

Table 20: **Crossword search method ablation.** All methods use one candidate, 500 updates, step size 0.1, and the same trained energy.

Epochs	Letter (%)	Word (%)
250	62.60	27.50
500	75.80	44.00
750	79.00	48.50
1,000	79.00	48.50
1,250	77.60	48.00
1,500	79.20	**51.00**
1,750	79.00	49.50
2,000	**79.40**	50.50

Table 21: **Crossword training duration ablation.** A separate sweep using GD, one candidate, 500 updates, step size 0.1, $\lambda = 8$, weight decay 0.8, and standard model weights.

Inference Budget. Table 22 varies candidate count and refinement steps separately. With 500 updates, increasing the candidate count from one to 1,024 changes letter accuracy from 80.4% to 80.8%, while word accuracy decreases from 51.0% to 50.5%. With 512 candidates, both 100 and 500 updates achieve 81.0% letter accuracy and 50.5% word accuracy. Thus, additional candidates or updates

provide limited benefit under these settings. The 100 update result uses 512 candidates and should not be interpreted as a one-candidate result.

Candidate-count sweep			Refinement-step sweep		
Candidates	Letter (%)	Word (%)	Updates	Letter (%)	Word (%)
1	80.40	**51.00**	100	**81.00**	**50.50**
128	**81.00**	50.00	200	80.80	50.00
512	**81.00**	50.50	300	80.80	50.00
1,024	80.80	50.50	400	**81.00**	**50.50**
–	–	–	500	**81.00**	**50.50**
–	–	–	1,000	80.80	50.00

Table 22: **Crossword inference budget ablations.** Both sweeps use GD, $\lambda = 8$, step size 0.1, standard model weights, and no step-size decay. The candidate-count sweep fixes 500 updates. The refinement-step sweep fixes 512 candidates.

Target Scale and Weight Decay. Table 23 reports separate target scale and weight decay sweeps. Among the tested values, $\lambda = 8$ and weight decay 0.8 achieve the highest letter and word accuracy within their respective sweeps. These results show that training settings affect the quality of the learned objective even when the inference procedure remains unchanged.

Target scale sweep			Weight decay sweep		
λ	Letter (%)	Word (%)	Weight decay	Letter (%)	Word (%)
4	76.20	42.00	0.6	77.60	44.50
8	**79.20**	**51.00**	0.7	77.40	45.00
12	77.40	47.00	0.8	**79.20**	**51.00**
–	–	–	0.9	77.80	45.50
–	–	–	1.0	77.20	44.00

Table 23: **Crossword training setting ablations.** Both sweeps use 1,500 training epochs and evaluate with GD, one candidate, 500 updates, step size 0.1, and standard model weights. The target scale sweep fixes weight decay 0.8. The weight decay sweep fixes $\lambda = 8$.

EMA and Step-Size Decay. In a separate evaluation with GD, 512 candidates, 100 updates, and initial step size 0.1, standard model weights achieve 81.0% letter accuracy and 50.5% word accuracy, compared with 44.4% and 8.0% using EMA weights. Applying step-size decay yields 80.8% letter accuracy and 50.0% word accuracy. The reported main configuration therefore uses standard model weights and no step-size decay.

Qualitative Example. The main-paper refinement figure shows a selected test puzzle evaluated with one candidate, 500 GD updates, and step size 0.1. It recovers 23 of 25 letters and seven of ten complete words. The two remaining letter errors affect three intersecting entries, illustrating why high letter accuracy need not translate into equally high word accuracy. This selected example illustrates refinement behavior rather than average benchmark performance.

C Planning Experiments

C.1 Shortest Path Finding

Data and Evaluation. We follow the directed-graph shortest-path benchmark of Du et al. [17]. Each instance provides a graph adjacency matrix, a start node, and a goal node, and the model predicts a sequence of node visits. Training uses 90,000 graphs with 15 nodes. Evaluation uses 10,000 held-out 15-node graphs from a similar distribution and 10,000 larger graphs with 25 nodes.

Following the baseline evaluation protocol, we score the first predicted action. An action is correct if it follows a valid directed edge and decreases the shortest-path distance to the goal. Reported accuracy therefore measures the quality of the next action, rather than successful execution of an entire predicted trajectory.

Model and Training. EMS uses the spatial-temporal graph convolutional backbone of IRED [17], with diffusion-time conditioning removed. The network contains two stacked `NLMConv1DBlockV2`

blocks with hidden width 64, followed by a per-node, per-step linear projection. The resulting output field $\mathbf{z}_\theta(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{T \times N}$ defines the trajectory energy through the parameterization described in Appendix A.1, where T is the prediction horizon and N is the number of graph nodes.

Training uses batch size 2,048, learning rate 10^{-4}, and 1.3×10^6 training updates. The main configuration uses $\lambda = 8$, contrastive weight $\beta = 0.5$, and linear target decay $c(\gamma) = 1 - \gamma$. We maintain EMA weights with decay 0.995 and use them for evaluation.

Unlike the local training used in the reasoning experiments, this model learns an energy over complete predicted trajectories conditioned on the graph and problem instance. The graph backbone supports variable numbers of nodes, allowing the same trained parameters to be evaluated on both 15-node and 25-node graphs.

Main Inference Configuration. The main comparison uses eight independently initialized candidates and ULA+GD refinement. Each candidate undergoes 200 ULA updates followed by 30 GD updates, with step size 0.05 in both phases. The graph, start node, goal node, and learned energy remain fixed throughout refinement. We select the candidate with the lowest final energy and decode its first action by taking the argmax over the corresponding node scores.

This configuration achieves 96.3% first-action accuracy on 15-node graphs and 93.1% on 25-node graphs. The baseline results for Diffusion, IREM, and IRED are reproduced from Du et al. [17]. Appendix A.3 reports model-evaluation counts, including the additional cost of searching over eight candidates.

Search Methods. Table 6 in the main paper compares search methods on the trained EMS energy. ULA+GD achieves the highest accuracy on both evaluation distributions, while GD alone remains competitive at 93.1% and 92.5%. The remaining methods achieve lower accuracy under their evaluated configurations. These results motivate combining initial stochastic exploration with deterministic refinement for the main comparison, without implying that stochastic methods uniformly improve planning performance.

Inference Budget. Tables 24 and 25 report separate refinement step and candidate count sweeps. The refinement-step sweep uses ULA with one candidate, $\lambda = 8$, and step size 0.05. Increasing refinement from 25 to 400 updates improves similar distribution accuracy from 87.2% to 93.0%. Larger graph accuracy reaches its highest observed value at 200 updates and decreases slightly at 400.

In the candidate count sweep, increasing the population from one to eight improves accuracy from 90.8% to 96.3% on similar graphs and from 86.9% to 93.1% on larger graphs. Increasing to 16 candidates provides a further 0.2 percentage point improvement on larger graphs, while similar distribution accuracy decreases. Additional inference computation therefore does not produce uniformly increasing accuracy. The step sweep uses ULA alone and is distinct from the main ULA+GD configuration.

Updates	Similar (%)	Larger (%)
25	87.2	80.9
50	90.5	80.0
100	91.8	82.5
200	92.4	**84.1**
400	**93.0**	83.3

Table 24: **Shortest-path refinement-step ablation.** First action accuracy using ULA, $\lambda = 8$, one candidate, and step size 0.05. Similar and Larger denote 15-node and 25-node evaluation graphs, respectively.

Candidates	Similar (%)	Larger (%)
1	90.8	86.9
2	94.0	90.3
4	95.2	92.1
8	**96.3**	93.1
16	95.3	**93.3**

Table 25: **Shortest-path candidate-count ablation.** First action accuracy when selecting the lowest energy final candidate from each population.

Contrastive Supervision. Table 26 reports the separate contrastive-supervision ablation. Adding the contrastive term increases larger-graph accuracy from 82.7% to 86.9%, while reducing similar-distribution accuracy from 92.5% to 90.8%. Consequently, the accuracy gap between the two distributions decreases from 9.8 to 3.9 percentage points. This ablation shows a benefit on larger graphs under the evaluated settings, rather than a uniform improvement across both distributions.

Qualitative Interpretation. Figure 9 illustrates changes in the decoded first action during refinement. Intermediate candidates can select an invalid edge or an action that does not reduce distance to the goal. The final illustrated candidate selects a valid shortest-path action. This example visualizes

Contrastive supervision	Similar (%)	Larger (%)	Gap (pp)
Off	**92.5**	82.7	9.8
On ($\beta = 0.5$)	90.8	**86.9**	3.9

Table 26: **Shortest-path contrastive ablation.** First-action accuracy on similar and larger graphs. Gap is similar-distribution accuracy minus larger-distribution accuracy, in percentage points.

the first-action metric used in evaluation and does not establish that every subsequent action in the predicted sequence is valid.

C.2 Maze Solving

Data and Evaluation. We evaluate long horizon planning on the Ghugare PointMaze suite [19]. Given start and goal configurations, the planner generates a fixed horizon waypoint trajectory that is executed with a low level inverse-dynamics controller. Success is determined by the environment's goal-reaching predicate, with episodes allowing up to 1,000 environment steps. We report mean success over five random seeds on Medium and Large mazes.

We train separate models using the corresponding trajectory datasets, with prediction horizons of 528 waypoints for Medium and 736 for Large. These experiments learn from complete trajectories. Short-segment training and trajectory stitching are evaluated separately in Appendix C.3.

Model and Training. EMS uses the one-dimensional U-Net backbone of Decision Diffuser [2], with diffusion-time conditioning removed. The network has base channel width 96, channel multipliers $(1, 2, 4, 8)$, residual blocks with kernel size 5 and Mish activations, and start–goal inpainting token dimension 32. Its output defines a scalar trajectory energy through the parameterization described in Appendix A.1.

Training uses batch size 512, learning rate 4×10^{-4}, and 5×10^5 training updates. We maintain EMA weights with decay 0.995 and use them for evaluation. The main configurations use $\lambda = 0.5$ for Medium and $\lambda = 2$ for Large, with linear target decay $c(\gamma) = 1 - \gamma$. Both environments use EqM alone ($\beta = 0$), without contrastive supervision or construction of contrastive negatives.

Inference and Baseline. We compare against Decision Diffuser [2], which generates trajectories through time-conditioned denoising. EMS instead refines candidate trajectories through gradient-based search on a fixed energy landscape.

The main EMS configurations use one candidate with 50 refinement updates on Medium and 75 on Large. Decision Diffuser uses one candidate and 512 denoising timesteps in both environments. These counts refer to individual plan generation calls. Appendix A.3 provides the corresponding budget definitions and accounting.

Quantitative Results. EMS achieves 33.67% mean success on Medium and 8.86% on Large, compared with 30.00% and 0.00% for Decision Diffuser, respectively (Figure 10). Although EMS obtains higher mean success in both environments, absolute success remains low on Large. The separate ablations below examine the effects of search method, target scale, and inference step size.

Search Methods. Table 27 compares search methods using $\lambda = 0.5$ on Medium and $\lambda = 2$ on Large. ULA performs best on Medium, reaching 36.67% success, whereas GD performs best on Large at 8.57%. ULA+GD obtains 28.33% and 2.86%, respectively. The preferred search method therefore differs between environments, motivating task-specific search settings within the shared energy-minimization formulation.

Inference Step Size. Table 28 varies the inference step size using ULA+GD on Medium and GD on Large. Medium achieves its highest observed success at $\eta = 0.0125$, while Large performs best at $\eta = 0.1$. Increasing the step size decreases Medium success over the tested range but improves Large success. These results show that the refinement step size must be considered alongside the learned energy and search method.

EqM Target Scale. Table 29 varies the EqM target gradient scale λ. Medium performs best at $\lambda = 0.5$, while Large performs best at $\lambda = 2$ among the tested values. Together with the search method and step size ablations, these results show that a shared training objective does not eliminate the need to tune training and inference settings for each environment.

Method	Medium (%)	Large (%)
GD	6.67	**8.57**
NAG	6.67	0.00
ULA	**36.67**	2.86
MALA	18.33	5.71
ULA+GD	28.33	2.86

Table 27: **PointMaze search method ablation.** Success rates using $\lambda = 0.5$ on Medium and $\lambda = 2$ on Large.

η	Medium (%)	Large (%)
0.0125	**33.33**	–
0.025	31.67	5.71
0.05	28.33	8.57
0.10	16.67	**10.00**

Table 28: **PointMaze step size ablation.** Medium uses ULA+GD with $\lambda = 0.5$. Large uses GD with $\lambda = 2$.

λ	Medium (%)	Large (%)
0.5	**28.33**	0.00
1.0	5.00	0.00
2.0	8.33	**2.86**

Table 29: **PointMaze target scale ablation.** Success rates as the EqM target gradient scale varies.

Qualitative Illustration. Figure 11 schematically illustrates how candidate waypoints are refined toward a start-to-goal trajectory. The learned trajectory energy guides the updates, while the start and goal specify the planning problem. The low success rate on Large indicates that this refinement procedure still struggles with the longer and more constrained trajectories in that environment.

C.3 Compositional Planning

Setup. We investigate long-horizon planning by training EMS on short trajectory segments and stitching them at inference under the CompDiffuser protocol [39]. We evaluate on the Medium, Large, and Giant environments of the OGBench PointMaze `stitch` suite. Unlike the full-instance experiments in Appendix C.2, training examples contain short segments rather than complete long-horizon plans.

EMS replaces diffusion-based segment refinement with optimization of learned time-independent energies within the stitching pipeline. Neighboring segments provide context for local refinement, and the resulting segments must agree where they overlap to form a coherent trajectory. We compare against the CompDiffuser results reported by Luo et al. [39].

Quantitative Results. Table 30 reports the original stitching results. EMS matches CompDiffuser at 100% success on Medium, but achieves 50% versus 100% on Large and 0% versus 68% on Giant. Thus, successful composition on Medium does not extend reliably to the larger environments.

For Medium, the reported EMS configuration uses 25 refinement updates and five candidates, compared with 512 denoising timesteps and 40 candidates for CompDiffuser. These are step and candidate counts for the corresponding stitching configurations, rather than complete episode level computation measurements.

Segment Coordination. During training, neighboring segments come from the same trajectory and therefore provide mutually consistent context. At inference, separately generated segments can disagree at their boundaries. Local refinement must then reconcile combinations of neighboring segments that were not encountered during training.

To address this mismatch, we corrupt neighboring segment context with independent noise during training. On Large, this reduces seam mismatch from 3.4 to 1.7 maze cells and increases the followable fraction of the plan from 86% to 94%. Goal reaching success improves from 50% to 70% (Table 31). These results indicate that exposing the model to inconsistent context improves segment coordination, although a substantial gap to CompDiffuser remains.

Remaining Failure on Giant. Context corruption does not resolve the Giant-maze failure. One possible explanation is the use of maze-normalized coordinates. As maze size increases while corridor width remains fixed in maze cells, the same normalized position error corresponds to a larger displacement and can move a waypoint across a wall.

The training trajectories do not contain wall crossing examples, so valid trajectory supervision alone does not explicitly teach the energy to penalize these errors. Improving segment coordination

Maze	CompDiffuser	EMS
Medium	$\mathbf{100} \pm 0$	**100**
Large	$\mathbf{100} \pm 0$	50
Giant	$\mathbf{68} \pm 3$	0

Table 30: **Compositional PointMaze planning.** Success rates (%) under the original stitching configuration. CompDiffuser values and uncertainties are reproduced from Luo et al. [39].

Metric	Original	Corrupted context
Success (%)	50	**70**
Seam mismatch (cells)	3.4	**1.7**
Followable plan (%)	86	**94**

Table 31: **Context corruption ablation on Large.** Corrupting neighboring-segment context during training improves coordination and execution. Lower seam mismatch and higher success and followable-plan fractions are better.

therefore does not by itself ensure collision-free plans. Training that directly addresses wall crossing candidates remains a direction for future work.

Discussion. These experiments show both the potential and the limits of applying the shared EMS formulation to trajectory stitching. The method succeeds on Medium, and a targeted training change improves Large maze performance. However, local segment quality and boundary coordination remain insufficient for reliable planning on Giant. Compositional long-horizon planning therefore remains a limitation of the current approach.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction describe EMS as a static energy-based framework for reasoning and planning, and the experiments support these claims across the evaluated reasoning and planning tasks.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 discusses inference costs and the difficulty of composing reliable long-horizon plans. Appendices B.2 and B.3 explain the separate N-Queens models by board size, the limited 3-SAT evaluation, and the lack of a matched full-formula fine-tuning comparison. Appendix C.3 reports compositional planning results on larger mazes, including the partial improvement from context corruption and failure on Giant.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper proposes and empirically evaluates a method. It does not present formal theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides implementation details, hyperparameters, architectures, datasets, evaluation protocols, and ablations in the appendix. We also include code with a README describing installation, data placement, and training/evaluation commands for reproducing the main experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide code with a README containing installation instructions, dataset download locations, expected data directory structure, and training/evaluation commands. The experiments use publicly available datasets, and the appendix reports implementation details and hyperparameters needed to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text describes the tasks, evaluation metrics, and primary comparisons, while the appendix provides full implementation details including data splits, architectures, hyperparameters, sampler settings, particles, and sampling steps. The README also gives data setup and training/evaluation instructions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Graph-coloring results report mean and standard deviation across five graphs per distribution. PointMaze results report mean and standard deviation across five random seeds. The main 3-SAT results report mean and standard error across three seeds, with standard error computed as the sample standard deviation divided by $\sqrt{3}$. N-Queens results report successful solutions out of 100 trials. Separate single-run ablations are identified as such.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: We report training and inference configurations and model-evaluation budgets, but do not provide a complete accounting of hardware resources, memory requirements, and execution times for every experiment.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research uses standard public benchmark datasets for reasoning and planning and does not involve human subjects, private data, or deployment in high-risk settings. We have reviewed the NeurIPS Code of Ethics and are not aware of any deviations.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is foundational research on search-based reasoning and planning algorithms using public benchmark tasks, and it is not tied to a specific deployment domain or direct societal application. We do not identify a direct path to societal impact beyond general improvements to optimization and planning methods.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models, scraped datasets, or systems with clear dual-use misuse risks such as language models, image generators, or surveillance models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the original papers and repositories for the datasets and baselines used in the experiments, and the released README documents/links the external assets that contain their license and terms-of-use information.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release code for training and evaluation, along with a README documenting installation, data preparation, expected directory structure, and commands for running experiments. The paper does not introduce a new dataset. It only uses existing public benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development and experiments do not use LLMs as an important, original, or non-standard component. Any LLM usage was limited to writing, editing, or formatting assistance and did not affect the methodology, scientific rigor, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.