CUTWORLD: A LATENT DYNAMICS MODEL FOR UNIFIED BRANCH-AND-CUT CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Branch-and-Cut solvers repeatedly choose a branching variable, a cutting plane, and an open search node. These choices reshape an exponentially large search tree, but standard learned branching rules score candidates only from the current relaxation. We present CUTWORLD, a latent decision-dynamics model that plans over candidate branching actions without solving their child linear programs. A bipartite GATv2 encoder maps the current relaxation to a global latent and per-variable embeddings. A causal Transformer then predicts the latent successor and its variable embeddings conditioned on a candidate and branch direction, allowing the policy to be re-applied to imagined states. We combine the resulting rollout value with the predicted dual-bound progress and cost-to-go, and use the shared representation to order nodes and select the root Gomory cut. Because the learned components only select among valid branches, globally valid cuts, and open-node orderings, they do not alter feasibility or bound tests. On fixed Set Cover benchmarks generated with one seed, the neural configurations complete all 10 medium instances (500×1000, 300 s limit) and all 5 hard instances (1000×2000, 600 s limit) in the Python/HiGHS harness, whereas the classical rules reach their time limits. A 3-step latent rollout reduces the medium-tier shifted-geometric-mean tree-node count by 44.2% relative to imitation (3,197 versus 5,733); at hard scale, combining rollout and root cuts reaches 2,603 nodes versus 4,633 for imitation. We report completion, tree-node counts, wall-clock time, LP time, and cuts, and explicitly separate completed-tree measurements from timeout lower bounds. The benchmark is intended as a controlled feasibility study; its small, single-seed evaluation does not establish statistical superiority over industrial MIP solvers.

1 INTRODUCTION

Mixed-Integer Programming (MIP) solvers are sequential decision systems. Within one solve, a Branch-and-Cut (B&C) engine repeatedly decides which fractional variable to branch on, which valid inequalities to add, and which open node to process next. A decision changes the relaxations encountered by all later decisions, so the quality of a rule is determined by the search tree it induces rather than by the current node alone.

This structure is a natural setting for model-based learning. In sequential decision problems, MuZero and Dreamer learn a latent transition model and plan by evaluating imagined futures (Schrittwieser et al., 2020; Hafner et al., 2020). The analogous operation in MIP is expensive: strong branching estimates the effect of a candidate by tentatively solving child LPs, and cut selection can evaluate many candidate inequalities. Learned MIP policies amortize strong branching with a graph network (Khalil et al., 2016; Gasse et al., 2019), but usually remain myopic at inference time. They predict the expert's choice from the current relaxation without learning a forward model of how the choice changes the next relaxation.

We introduce CUTWORLD, a first realization of latent decision dynamics inside B&C. The model has three properties. First, it predicts both a graph-level successor latent and per-variable successor embeddings; the latter are essential because the branching policy must be re-evaluated on an imagined child. Second, it grounds the latent with dual-bound and reward heads and predicts cost-to-go, giving the rollout a search-specific objective rather than a generic representation loss. Third, it reuses

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

the latent for branching, open-node ordering, and root-cut selection, while keeping all exact-solver tests outside the learned model.

Contributions.

- We formulate learned B&C control as latent decision dynamics and introduce a per-variable transition model that supports policy reapplication in imagined states.
- We derive a search-aware rollout objective that combines predicted dual-bound progress, value bootstrap, and cost-to-go, and provide a correctness argument for using learned decisions within exact tree search.
- We implement shared latent control of branching, node ordering, and root Gomory-cut selection in a single Python/HiGHS research harness.
- We provide a controlled Set Cover study with fixed instances, explicit timeout semantics, shifted geometric means, a component ablation, and measurements of solver and model times. The results identify both the value of shallow imagination and the cost of inaccurate long-horizon dynamics.

2 Related Work

Learning to branch. Early work learned branching scores from hand-crafted MIP features (Khalil et al., 2016). Gasse et al. (Gasse et al., 2019) introduced a bipartite graph representation of variables and constraints and trained a graph convolutional policy to imitate strong branching. This representation is the starting point for our encoder, while our central difference is a learned transition model that is used after training for prospective lookahead. Tree-structured returns (Scavuzzo et al., 2022) and neural diving and branching (Nair et al., 2020) demonstrate the value of longer-horizon objectives, but do not model the solver transition explicitly. The broader methodological landscape is surveyed by Bengio et al. (2021).

Learning to cut and optimize. Cut selection has been formulated as reinforcement learning (Tang et al., 2020), imitation from cut lookahead (Paulus et al., 2022), and hierarchical sequence prediction (Wang et al., 2023). These approaches motivate our use of candidate-level cut features, but our deployment restricts the learner to selecting from a pool of validity-certified Gomory cuts. Learned large neighborhood search provides another example of learned solver control (Sonnerat et al., 2021). We focus instead on exact tree search, where learned decisions must remain separate from pruning and feasibility logic.

Sequence models and model-based planning. Mamba-Branching models realized branching histories with a state-space model (Zhang et al., 2026). Such retrospective history modeling complements our prospective model: we predict the consequences of an unrealized action before committing to it. CutWorld follows the latent planning principle of MuZero and Dreamer (Schrittwieser et al., 2020; Hafner et al., 2020), but the state is a constraint–variable relaxation and the action space is solver-specific.

3 Background and Problem Formulation

3.1 Mixed-Integer Programming and Branch-and-Cut

We study binary MIPs of the form

$$\min_{x\in\{0,1\}^n} c^\top x \quad \text{s.t.} \quad Ax \geq b, \qquad A \in \mathbb{R}^{m\times n},\ c \in \mathbb{R}^n,\ b \in \mathbb{R}^m. \tag{1}$$

Set Cover is the special case $A \in \{0,1\}^{m\times n}$ and $b = \mathbf{1}$. At a search node u, the integer restrictions are relaxed and the solver obtains

$$\ell(u) = \min_{x\in[0,1]^n} c^\top x \quad \text{s.t.} \quad A_u x \geq b_u. \tag{2}$$

If $x_u^\star$ is fractional, branching on j creates two descendants with $x_j \le 0$ and $x_j \ge 1$ (equivalently, $x_j = 0$ and $x_j = 1$ for binary variables). A valid cut $\alpha^\top x \ge \beta$ tightens the LP relaxation without removing any integer-feasible point.

For a node with incumbent objective U and dual bound $\ell(u)$, the relative primal–dual gap is

$$g(u) = \frac{|U - \ell(u)|}{\max\{1, |U|\}}. \tag{3}$$

The learned policy never computes $g(u)$ and never changes the bound, integrality, or feasibility tests. It only selects actions and open-node orderings.

3.2 A Latent Decision Process

We represent a search state s_t by the current relaxation graph, its LP solution, and its root-to-node decision history. The action is $a_t = (v_t, d_t)$, where v_t is a fractional variable and $d_t \in \{-1, +1\}$ is the branch direction. Cuts are treated as a validity-certified auxiliary action at the root. The solver induces a transition $P(s_{t+1} \mid s_t, a_t)$ by solving the child relaxation.

The objective is to reduce search effort while preserving correctness. We use dual-bound progress as a dense auxiliary reward,

$$r_t = \mathrm{norm}(\ell(s_{t+1}) - \ell(s_t)), \qquad J = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t - \eta \log(1 + N_{\mathrm{tree}})\right], \tag{4}$$

where N_{tree} is the number of search-tree nodes processed and η controls the search-cost auxiliary. The true transition is expensive; CutWorld learns a latent approximation $\widehat{P}$.

Proposition 1 (Correctness under admissible control)**.** *Assume that (i) every branching variable is selected from the current fractional binary variables, (ii) every added cut is globally valid for the original integer-feasible set, and (iii) pruning is performed only from certified infeasibility, integrality, or valid bound comparisons. Then, replacing the baseline branching rule, cut ordering, or open-node ordering by any* CutWorld *policy preserves the set of optimal integer solutions returned by Branch-and-Cut.*

4 CutWorld

We view Branch-and-Cut as a sequential decision process. At each node t of the search, the solver observes a state s_t (the current relaxation), takes an action a_t (branch on a fractional variable, or add a cut), and transitions to the next node. We learn (i) an encoder that maps s_t to a latent state, (ii) a dynamics model that predicts how that latent evolves under an action, and (iii) heads that score decisions by rolling the dynamics forward (Figure 1). Throughout, $d = 128$ denotes the embedding dimension.

4.1 Bipartite Node Encoder

The relaxation is a bipartite graph $G_t = (\mathcal{V}_t, \mathcal{C}_t, \mathcal{E}_t)$ over variables and constraints. Each variable node contains objective, bound, LP solution, fractionality, reduced cost, and basis features; each constraint node contains normalized right-hand side, tightness, dual, and age features. Each edge contains its coefficient, a right-hand-side-normalized coefficient, and its sign. We use 19 variable features, 5 constraint features, and a three-dimensional edge feature.

Let $h_v^{(0)}$ and $h_i^{(0)}$ denote initial variable and constraint embeddings. A GATv2 layer computes attention coefficients

$$\alpha_{iv}^{(\ell)} = \frac{\exp\left(a^\top \mathrm{LeakyReLU}\left(W_c h_i^{(\ell)} + W_v h_v^{(\ell)} + W_e e_{iv}\right)\right)}{\sum_{k \in \mathcal{N}(v)} \exp\left(a^\top \mathrm{LeakyReLU}\left(W_c h_k^{(\ell)} + W_v h_v^{(\ell)} + W_e e_{kv}\right)\right)} \tag{5}$$

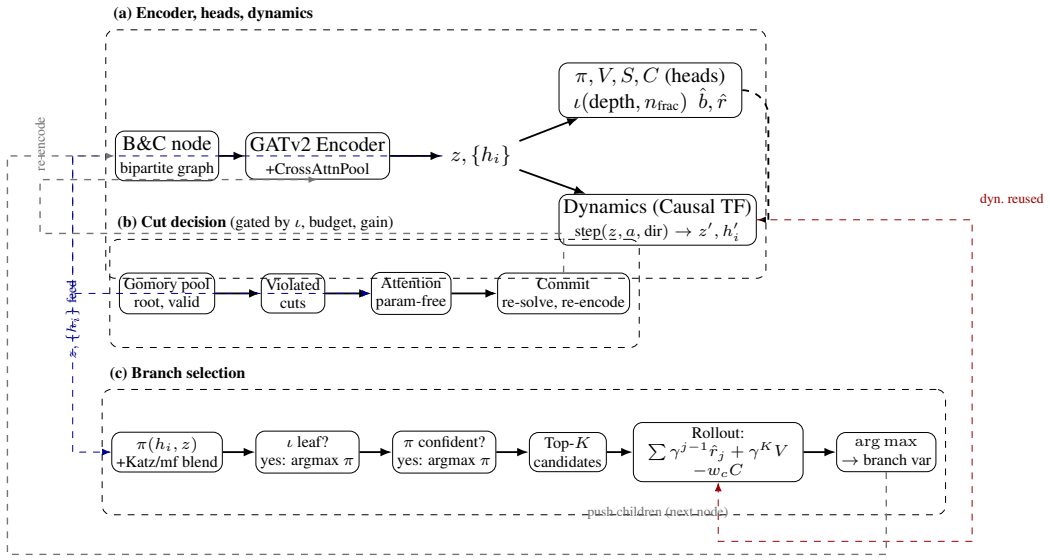


Figure 1: **CutWorld architecture and decision pipeline. (a)** A GATv2 encoder produces graph latent z and per-variable embeddings $\{h_i\}$; a causal Transformer predicts their evolution under candidate actions. **(b)** Valid Gomory cuts are selected by attention, subject to integrality, budget, gain, and depth checks, then committed with an LP re-solve. **(c)** Fractional variables are scored by π; confident nodes exit early, while ambiguous nodes use latent rollout with repeated policy re-application and reward/cost-to-go scoring. Dashed links indicate shared representations ($z, \{h_i\}$) and dynamics reuse; gray loops denote cut re-solving and branch-node expansion. Learned components only select valid actions and do not alter LP bounds or feasibility tests.

and updates the two node types by normalized aggregation:

$$\widetilde{h}_v^{(\ell+1)} = \sum_{i \in \mathcal{N}(v)} \alpha_{iv}^{(\ell)} W_c h_i^{(\ell)}, \qquad h_v^{(\ell+1)} = \mathrm{LN}\Big(h_v^{(\ell)} + \mathrm{MLP}_v(\widetilde{h}_v^{(\ell+1)})\Big), \tag{6}$$

$$\widetilde{h}_i^{(\ell+1)} = \sum_{v \in \mathcal{N}(i)} \alpha_{vi}^{(\ell)} W_v h_v^{(\ell)}, \qquad h_i^{(\ell+1)} = \mathrm{LN}\Big(h_i^{(\ell)} + \mathrm{MLP}_c(\widetilde{h}_i^{(\ell+1)})\Big). \tag{7}$$

The implementation uses three layers and four attention heads with $d = 128$. A learned query q pools the variable embeddings:

$$\omega_i = \mathrm{softmax}_i\left(\frac{(W_q q)^\top (W_k h_i)}{\sqrt{d}}\right), \qquad z_t = \sum_{i=1}^{n} \omega_i W_v h_i, \qquad \mathrm{Enc}(s_t) = (z_t, \{h_i\}_{i=1}^n). \tag{8}$$

Unlike a uniform mean, Eq. (8) can emphasize variables that are fractional, expensive, or structurally connected to tight constraints.

4.2 Latent Dynamics

The action embedding concatenates the selected variable representation and a direction code:

$$a_t = W_a\,[h_{v_t} \,\|\, \mathrm{onehot}(d_t)]. \tag{9}$$

The causal Transformer receives tokens

$$u_\tau = W_{\text{in}}[z_\tau \,\|\, a_\tau \,\|\, \mathrm{onehot}(d_\tau)] + p_\tau, \qquad \tau \le t, \tag{10}$$

where p_τ is a learned positional embedding. A four-layer, four-head Transformer predicts a residual successor:

$$\Delta z_{t+1} = W_z\ \mathrm{TF}_{\text{causal}}(u_{0:t}), \qquad \widehat{z}_{t+1} = \mathrm{LN}(z_t + \Delta z_{t+1}). \tag{11}$$

Algorithm 1 Policy-anchored latent branching

Require: Encoded state $(z, \{h_i\})$, fractional set $\mathcal{F}$, horizon K, candidate budget k, rollout weight λ

```
ℓ_i ← policy logits from Eq. (15)
if max_{i∈F} softmax(ℓ)_i ≥ θ then
    return arg max_{i∈F} ℓ_i
end if
C ← TopK_k(ℓ, F)
for c ∈ C do
    R̂(c) ← latent rollout from Eq. (17)
end for
Normalize ℓ_c and R̂(c) over C
return arg max_{c∈C} [zscore(ℓ_c) + λ zscore(R̂(c))]
```

The per-variable transition is

$$\widehat{h}_i^{t+1} = \mathrm{LN}\big(h_i^t + \mathrm{MLP}_h[h_i^t \,\|\, \widehat{z}_{t+1} \,\|\, a_t]\big)\,. \tag{12}$$

This factorization is important: a graph-level prediction alone cannot re-score the candidate variables at the imagined state.

Two grounding heads connect the latent to solver quantities:

$$\widehat{\ell}_{t+1} = w_\ell^\top \widehat{z}_{t+1}, \qquad \widehat{r}_t = w_r^\top \widehat{z}_{t+1}. \tag{13}$$

4.3 Policy, Value, and Cost-to-Go Heads

Let $\bar{h}_{\mathrm{frac}}$ be the mean embedding of currently fractional variables. The value and cost-to-go heads use this state-dependent summary:

$$V(s_t) = f_V([z_t \,\|\, \bar{h}_{\mathrm{frac}}]), \qquad C(s_t) = \mathrm{softplus}\left(f_C([z_t \,\|\, \bar{h}_{\mathrm{frac}}])\right). \tag{14}$$

The target for C is $\log(1+T-t)$, where $T-t$ is the number of remaining transitions in a trajectory. The policy is a pointer scorer over $i \in \mathcal{F}_t$:

$$k_i = W_k h_i, \qquad \ell_i = \frac{w_\pi^\top \tanh(k_i + W_z z_t)}{\sqrt{d}}, \qquad \pi(i \mid s_t) = \mathrm{softmax}_{i\in\mathcal{F}_t}(\ell_i). \tag{15}$$

An integrality head predicts whether the current node is close to terminal:

$$p_{\mathrm{leaf}}(s_t) = \sigma(f_{\mathrm{int}}[z_t \,\|\, \mathrm{depth}(t)/D \,\|\, \log(1+n_{\mathrm{frac}})])\,. \tag{16}$$

It is used only as a computational gate; it cannot certify a node or trigger unsafe pruning.

4.4 Latent Rollout and Node Ordering

For a candidate c, the model takes $a_0 = h_c$ and recursively predicts K imagined states. At each imagined state, the policy chooses the next action from the predicted fractional set. A depth-K score is

$$R_K(c) = \sum_{j=1}^{K} \gamma^{j-1} \widehat{r}_j + \gamma^K V(\widehat{s}_K) - w_C \sum_{j=1}^{K} \gamma^{j-1} \widehat{C}(\widehat{s}_j). \tag{17}$$

The terminal value is discounted once for each latent transition. For a candidate set $\mathcal{C}_t = \mathrm{TopK}_k(\pi, \mathcal{F}_t)$, we standardize policy logits and rollout scores within the candidate set and select

$$c^\star = \arg\max_{c\in\mathcal{C}_t} \left[\mathrm{zscore}(\ell_c) + \lambda\, \mathrm{zscore}(R_K(c))\right]. \tag{18}$$

When the policy is sufficiently confident, the rollout is skipped. Open nodes are ordered by predicted cost-to-go, with certified lower bounds retained as a secondary key.

4.5 ROOT GOMORY-CUT SELECTION

CUTWORLD selects cuts from a validity-certified root pool. Let a basic tableau row of the root LP be

$$x_B + \sum_{j \in \mathcal{N}} \bar{a}_j x_j = \bar{b}, \qquad x_B \notin \mathbb{Z}, \quad \bar{b} \notin \mathbb{Z}. \tag{19}$$

Writing $\mathrm{frac}(q) = q - \lfloor q \rfloor$, the Gomory fractional cut is

$$\sum_{j \in \mathcal{N}} \mathrm{frac}(\bar{a}_j) x_j \geq \mathrm{frac}(\bar{b}). \tag{20}$$

We express the cut in the original structural variables and retain it only when the generator's validity checks pass. A cut feature vector contains violation, efficacy, density, objective cutoff, support fraction, and parallelism. A parameter-free attention score is

$$q_\ell = \frac{(W_c \phi_\ell)^\top (W_z z_t)}{\sqrt{d}}, \qquad \rho_\ell = \mathrm{softmax}_\ell(q_\ell), \tag{21}$$

and the highest-scoring cut is committed, subject to the cut budget. This is a root cut selection, not a claim that a new separator is learned at every descendant. The distinction matters for both validity and computational cost.

5 EXPERIMENTS

5.1 BENCHMARK AND PROTOCOL

We generate binary Set Cover instances with density 0.05. The medium tier has 500 constraints and $1{,}000$ variables with a 300 s limit; the hard tier has $1{,}000$ constraints and $2{,}000$ variables with a 600 s limit. The reported benchmark contains 10 medium and 5 hard instances generated with seed 1. All methods use the same Python/HiGHS search harness, node limits, and instance order. The model checkpoint is `checkpoints/highs_retrain/phase4_final.pt`.

The primary neural configuration uses a top-$k = 3$ candidate set, both branch directions, $\gamma = 0.95$, cost-to-go weight 1.0, attention-based root cut selection, and the primal heuristic configured by the harness. A 3-step latent rollout ($K = 3$) is the evaluated configuration. The size penalty is reported separately because its training scale differs from the medium and hard evaluation scales.

Baselines. We compare most-fractional branching with strong-branching probe budgets 1, 4, and 8 (SB-1/4/8), imitation-only branching, a policy/most-fractional blend, root-cut selection, latent rollout, and the combined neural policy. The classical rows are LP-based controls in the same research harness; they are not a replacement for a separately tuned default SCIP baseline.

5.2 METRICS AND TIMEOUT SEMANTICS

We distinguish a completed search from an independently verified optimum. Let N_{tree} be the number of main search-tree LP nodes processed by the harness and let N_{LP} include additional strong-branching probe LPs. The tables report N_{tree}; the probe counts are retained in the raw JSON results because SB can incur substantial computational cost without adding a main tree node.

For positive measurements $y_1, \ldots, y_n$, the shifted geometric mean is

$$\mathrm{SGM}_\delta(y) = \exp\left(\frac{1}{n} \sum_{i=1}^{n} \log(y_i + \delta)\right) - \delta. \tag{22}$$

We use $\delta_N = 10$ for tree nodes and $\delta_T = 1$ second for wall time. Timeout rows contribute their time limit and their partial tree-node count; therefore, a timeout node count is a lower bound on the completed tree. The "relative incumbent gap" is

$$\Delta_{\mathrm{inc}} = 100 \, \frac{z - \min_{m'} z_{m'}}{\max\{1, |\min_{m'} z_{m'}|\}}, \tag{23}$$

where the minimum is taken over feasible incumbents observed within the same benchmark budget. It is not a proof gap to a known optimum.

Table 1: Medium tier: 500×1000 Set Cover, 10 instances, 300 s limit. SGM uses shifts $\delta_N = 10$ and $\delta_T = 1$. "At limit" is a partial tree-node count and is not comparable with a completed-tree count. The gap is relative to the best feasible incumbent observed within the same budget, not an optimality certificate.

Method	Complete	SGM N_{tree} at limit	SGM N_{tree} at completion	SGM time (s)	Incumbent gap
Most-fractional	0%	4,873	–	301.4	1.2%
SB-1	0%	4,605	–	301.3	**1.1%**
SB-4	0%	3,024	–	301.0	1.5%
SB-8	0%	**2,710**	–	**300.8**	2.0%
Policy (imitation)	100%	–	5,733	237.6	4.3%
MF-blend	100%	–	4,165	**218.1**	1.6%
Policy + root cuts	100%	–	5,396	240.4	2.8%
Rollout	100%	–	**3,197**	271.4	5.2%
Rollout + size	100%	–	4,320	258.8	3.5%
Neural-Best	100%	–	3,235	242.4	**1.9%**

Table 2: Hard tier: 1000×2000 Set Cover, 5 instances, 600 s limit. Classical rows are partial trees at timeout; neural rows complete the harness search.

Method	Complete	SGM N_{tree} at limit	SGM N_{tree} at completion	SGM time (s)	Incumbent gap
Most-fractional	0%	3,342	—	601.0	0.6%
SB-1	0%	2,631	—	600.9	0.7%
SB-4	0%	**1,074**	—	**600.4**	1.2%
SB-8	0%	1,096	—	600.5	1.8%
Policy (imitation)	100%	—	4,633	389.3	2.2%
MF-blend	100%	—	2,736	**353.1**	1.0%
Policy + root cuts	100%	—	4,419	395.2	2.8%
Rollout	100%	—	4,007	422.8	3.2%
Rollout + size	100%	—	4,376	396.3	2.7%
Neural-Best	100%	—	**2,603**	364.8	**1.4%**

5.3 Main Results

Table 1 reports the medium tier. Neural rows report completed trees; classical rows reach the time limit and therefore have no completed-tree SGM. The result is a sharp difference in completion rates in this particular small benchmark, but it should not be mistaken for evidence that the neural policy dominates a production MIP solver.

Shallow imagination reduces tree size. The 3-step rollout reduces SGM tree nodes from 5,733 to 3,197, a 44.2% reduction relative to imitation. The combined Neural-Best configuration achieves a similar 3,235 nodes because root cuts add a small amount of additional work but do not substantially change the branching tree.

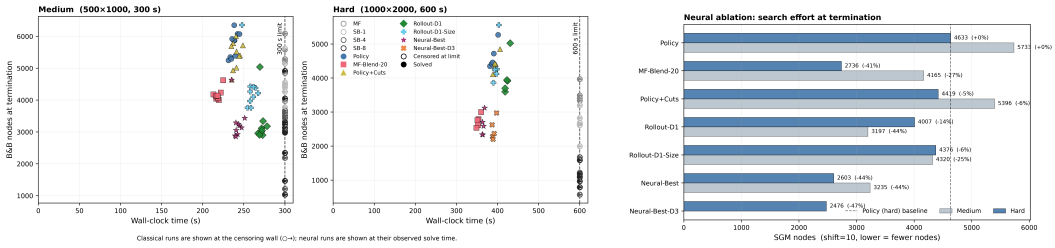
Wall time and tree size are different objectives. MF-blend is the fastest neural configuration at 218.1 s, while Rollout requires 271.4 s. The rollout saves tree nodes but spends additional model time. This distinction motivates reporting both N_{tree} and wall-clock time rather than selecting a checkpoint from node count alone.

5.4 Hard-Tier Scaling

Table 2 repeats the comparison at twice the dimensions and a 600 s budget. The combined Neural-Best configuration reaches the smallest completed-tree SGM among the neural configurations.

5.5 Cut Selection

Adding the attention-selected root cut pool to imitation reduces the number of medium-tier SGM nodes from 5,733 to 5,396 and the number of hard-tier nodes from 4,633 to 4,419. The effect is modest compared with rollout, but cuts and rollout address different mechanisms: cuts tighten the



(a) Tree nodes versus wall-clock time. Classical methods terminate at the time limit; their partial tree counts are lower bounds.

(b) Neural ablation using completed-tree SGM nodes.

Figure 2: Benchmark overview. The two panels should be interpreted with the timeout semantics in Section 5; completed neural trees and partial classical trees are not interchangeable observations.

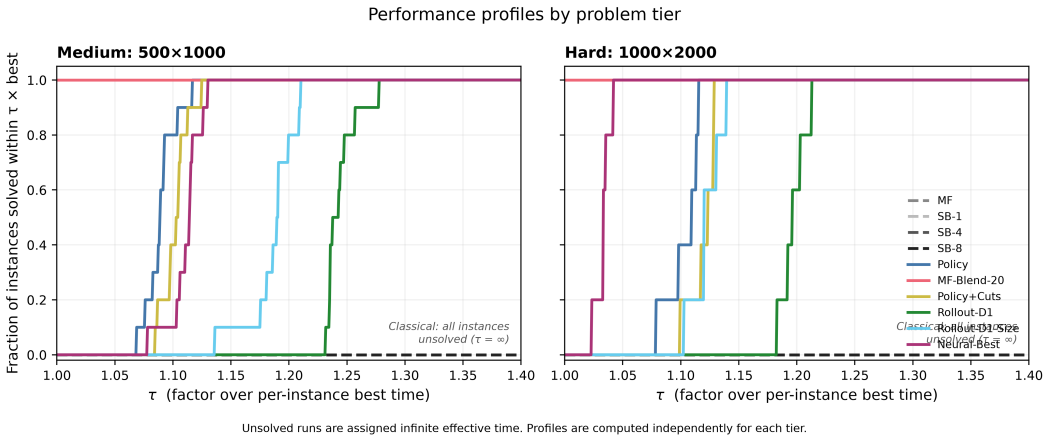


Figure 3: Performance profiles on medium and hard tiers. The profile reports completion within a factor of the best observed wall-clock time; timeout runs are assigned $\tau = \infty$.

root relaxation while rollout changes the branching order. Neural-Best combines both and reaches 3,235 medium-tier nodes and 2,603 hard-tier nodes. Because the reported cut experiment uses root-generated cuts, we interpret it as evidence for root cut selection rather than for a learned general-purpose separator.

6 Analysis and Limitations

The model reduces tree size but does not eliminate inference cost. The 3-step Rollout run reduces the completed search tree by 44.2% but has a wall time of 271.4 s, slower than MF-blend at 218.1 s. The raw result files, therefore, retain LP, encoding, policy, and rollout timers. This is the central engineering trade-off: latent planning is useful only when its per-node cost is amortized by sufficiently large tree reductions.

Completion is not a universal optimality comparison. The neural harness reports an empty open-node queue on every listed instance, whereas classical rows stop at the time limit. The evaluation files do not contain independently precomputed optima (`optima=null`), so "Complete" is the primary label in the tables. It should not be read as a claim that CutWorld has been shown to dominate a certified industrial solver. Future experiments should use larger held-out sets, multiple seeds, verified optima where feasible, primal–dual gaps at timeout, and an independently tuned SCIP baseline.

Scale and statistical power. There are 10 medium and 5 hard instances at a single generator seed. Consequently, the observed SGM differences are descriptive rather than statistically conclusive. Confidence intervals, paired tests, and memory measurements should be added before making claims about generalization.

Cut scope. The present cut pool is generated at the root. Applying the same pool throughout the tree is valid only because the cuts are globally valid; it is not equivalent to learning a node-specific separator. Extending the method to local cuts requires explicit bookkeeping of descendant-scoped validity.

7 CONCLUSION

We presented CUTWORLD, a latent decision-dynamics model for B&C control. The per-variable transition allows a branching policy to be reapplied to imagined states, while grounded reward, value, and cost-to-go heads turn latent prediction into a search-specific rollout objective. In a controlled Set Cover study, a 3-step latent rollout substantially reduces the completed-tree node count, but the additional inference cost means that MF-blend remains the fastest neural configuration.

The broader lesson is methodological: learned solver control should be evaluated as a multi-objective system. Tree nodes, LP work, wall time, completion status, incumbent quality, cut count, and memory can disagree. CUTWORLD provides a concrete architecture for studying this tension while keeping exact solver logic outside the learned components. Larger multi-seed benchmarks, verified optima, industrial-solver baselines, and memory-aware planning are the next steps toward a robust general-purpose world model for optimization.

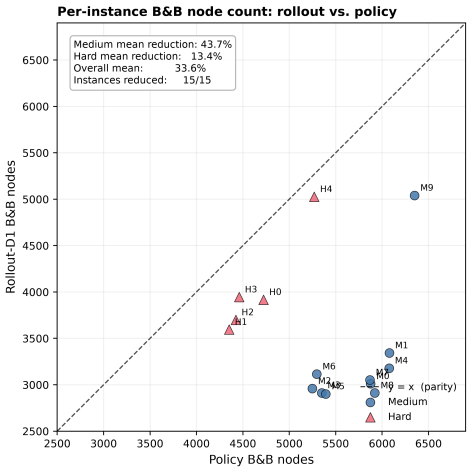


Figure 4: Per-instance tree-node comparison between Rollout and imitation. Points below the diagonal indicate fewer nodes under rollout.

AI USE STATEMENT

We used generative AI tools for limited language-related assistance, including grammar correction, proofreading, improving clarity and readability, and minor suggestions for wording and presentation. Generative AI tools were not used to generate the core technical ideas, methodology, experimental design, results, or conclusions of this work. All AI-assisted content was reviewed and edited by the authors, and the final manuscript was checked for correctness and consistency. The authors take full responsibility for the final content of the work, including all claims, text, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work presents a computational method for Branch-and-Cut optimization and evaluates it using synthetically generated Set Cover instances. The study does not involve human subjects, personal data, or sensitive data. The authors are not aware of any additional ethical concerns arising from the methods, data, or experiments presented in this work.

Reproducibility Statement

We have taken several steps to support reproducibility of our work. An anonymized version of the code, together with the instance-generation scripts, training scripts, and random seeds used in the experiments, is provided as supplementary material. The main paper describes the proposed method, experimental setup, and evaluation protocol, while the supplementary material provides the full hyperparameter configuration, data generation procedure, and additional implementation details. The computational environment used for the experiments is also reported to facilitate the reproduction of the results.

References

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence*, 2016.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Max B. Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris J. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International Conference on Machine Learning (ICML)*, 2022.

Lara Scavuzzo, Feng Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, et al. Mastering atari, go, chess and shogi by planning with a learned model. volume 588, pp. 604–609, 2020.

Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. In *arXiv preprint arXiv:2107.10201*, 2021.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning (ICML)*, 2020.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *International Conference on Learning Representations (ICLR)*, 2023.

Yiwei Zhang, Hao Zhang, and Jianfeng Fan. Mamba-branching: Sequence-aware variable branching for mixed-integer programming. In *International Conference on Learning Representations (ICLR)*, 2026.

A Appendix

A.1 Implementation Details

The encoder uses a hidden dimension $d = 128$, three GATv2 layers, and four attention heads. The latent dynamics model uses four causal Transformer layers, four attention heads, a feed-forward width of $4d = 512$, and a maximum sequence length of 512. The optimizer is AdamW with weight decay 10^{-4}, gradient clipping at 1.0, cosine learning-rate schedules, and automatic mixed precision on GPU.

Training follows a four-stage curriculum:

1. policy imitation;
2. value prediction;
3. latent dynamics and overshooting;
4. joint fine-tuning.

Each stage is warm-started from the preceding checkpoint. The reported checkpoint is `checkpoints/highs_retrain/phase4_final.pt`.

Table 3: Main model and solver hyperparameters.

Parameter	Value
Latent dimension	128
GNN layers / heads	3 / 4
Transformer layers / heads	4 / 4
Transformer FFN width	512
Maximum history length	512
Optimizer	AdamW
Weight decay	10^{-4}
Gradient clipping	1.0
Rollout candidate budget	$k = 3$
Rollout depth	$K = 3$
Discount factor	$\gamma = 0.95$
Branching factor	$b = 2$
Cost-to-go weight	1.0
Policy–fractionality blend	0.20
Size penalty	1.0 in size ablations

A.2 Feature Representation

Each search node is represented as a bipartite graph whose variable nodes, constraint nodes, and edges carry normalized solver features.

Table 4: Feature groups used by the bipartite encoder.

Element	Features
Variable node	objective coefficient, variable type, bounds, LP solution value, fractionality, reduced cost, basis status, incumbent statistics
Constraint node	normalized right-hand side, tightness, objective similarity, dual value, scaled age
Edge	coefficient, RHS-normalized coefficient, sign

Let x^{LP} denote the current LP solution. The fractionality feature for a binary variable is

$$f_i = \left| x_i^{\mathrm{LP}} - \mathrm{round}(x_i^{\mathrm{LP}}) \right|. \tag{24}$$

The candidate branching set is

$$\mathcal{F}_t = \{i : f_i > \varepsilon,\ f_i < 1 - \varepsilon\}, \qquad \varepsilon = 10^{-4}. \tag{25}$$

All continuous features are standardized using statistics computed only on the training split. Statistics are frozen during validation and testing.

A.3 Training Objectives

The policy imitates strong branching with a hard-label cross-entropy and a soft target derived from normalized expert scores:

$$\mathcal{L}_\pi = \mathrm{CE}(\pi, y_{\mathrm{SB}}) + \alpha\ \mathrm{KL}(\mathrm{softmax}(\widetilde{q}_{\mathrm{SB}}) \,\|\, \pi). \tag{26}$$

The value and cost-to-go losses use the Huber penalty

$$\mathcal{H}_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right), & |e| > \delta, \end{cases} \quad \mathcal{L}_V = \mathcal{H}_\delta(V - \bar{\ell}), \quad \mathcal{L}_C = \mathcal{H}_\delta(C - \log(1 + T - t)). \tag{27}$$

The one-step dynamics objective combines latent, variable, grounding, and reward terms:

$$\begin{aligned} \mathcal{L}_{\mathrm{dyn}} = {} & \|\widehat{z}_{t+1} - z_{t+1}\|_2^2 + \beta_h \frac{1}{n} \sum_i \|\widehat{h}_i^{t+1} - h_i^{t+1}\|_2^2 \\ & + \beta_\ell \mathcal{H}_\delta(\widehat{\ell}_{t+1} - \ell_{t+1}) + \beta_r \mathcal{H}_\delta(\widehat{r}_t - r_t) + \beta_{\cos}\left(1 - \cos(\widehat{z}_{t+1}, z_{t+1})\right). \end{aligned} \tag{28}$$

We additionally train on model-generated multi-step predictions:

$$\mathcal{L}_{\mathrm{over}} = \sum_{k=1}^{K_{\max}} \frac{1}{k} \|\widehat{z}_{t+k} - z_{t+k}\|_2^2. \tag{29}$$

The integrality objective is a weighted binary cross-entropy,

$$\mathcal{L}_{\mathrm{int}} = \mathrm{BCE}^{+}(p_{\mathrm{leaf}}, y_{\mathrm{leaf}}). \tag{30}$$

The joint objective used in the reported checkpoint is

$$\mathcal{L}_{\mathrm{joint}} = \mathcal{L}_\pi + 0.5\mathcal{L}_V + 0.1\mathcal{L}_{\mathrm{int}} + 0.1\mathcal{L}_{\mathrm{cons}} + 0.3\mathcal{L}_C + 0.5\mathcal{L}_{\mathrm{dyn}}, \tag{31}$$

where $\mathcal{L}_{\mathrm{cons}}$ aligns value predictions on real and imagined states. Training proceeds through the four stages listed above, warm-starting each stage from the previous checkpoint.

A.4 Evaluation Metrics

Let N_{tree} denote the number of main search-tree LP nodes processed by the solver. Let N_{LP} additionally include LPs solved for strong-branching probes. The main tables report N_{tree}, while N_{LP} is retained as an auxiliary computational metric.

For positive measurements $y_1, \ldots, y_n$, we use the shifted geometric mean

$$\mathrm{SGM}_\delta(y) = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \log(y_i + \delta)\right) - \delta. \tag{32}$$

The experiments use $\delta_N = 10$ for node counts and $\delta_T = 1$ second for wall-clock time.

A run is labeled *complete* when the open-node queue is exhausted before the time limit. A timeout count is a partial-tree count and is therefore a lower bound on the number of nodes required by a completed solve.

The relative incumbent gap reported in the tables is

$$\Delta_{\mathrm{inc}} = 100 \frac{z - \min_{m'} z_{m'}}{\max\{1, |\min_{m'} z_{m'}|\}}, \tag{33}$$

where the minimum is taken over feasible incumbents observed within the same benchmark budget. This quantity is not an independently verified optimality gap.

Table 5: Additional completed-tree ablation results.

Configuration	K	Cuts	Medium nodes	Hard nodes	Hard time
Policy	0	No	5,733	4,633	389.3
MF-blend	0	No	4,165	2,736	353.1
Policy + cuts	0	Yes	5,396	4,419	395.2
Rollout	3	No	3,197	4,007	422.8
Rollout + size	3	No	4,320	4,376	396.3
Neural-Best	3	Yes	3,235	2,603	364.8

A.5 Additional Ablation Results

The size penalty is not uniformly beneficial. It increases the medium-tier completed-tree SGM from 3,197 to 4,320 nodes and the hard-tier SGM from 2,603 to 4,376 nodes. This indicates that cost-to-go and subtree-size predictions extrapolate imperfectly from the training distribution.

A.6 Correctness Argument

Proposition 2 (Validity-preserving learned control)**.** *Suppose that every branching action is selected from the current fractional binary variables, every added cut is globally valid for the original integer program, and pruning is performed only using certified feasibility, integrality, and bound tests. Then, learned branching, cut-selection, and node-ordering policies preserve the set of optimal integer solutions.*

Proof. A branch on a fractional binary variable partitions the integer-feasible set into two disjoint subproblems whose union is the parent feasible set. A globally valid cut removes no integer-feasible point. Finally, changing the order of open nodes changes only the order in which subproblems are processed. Therefore, every node discarded by the solver remains certified infeasible, integer-feasible, or dominated by a valid bound comparison. When the queue is empty, the incumbent is consequently optimal under the same assumptions as the baseline Branch-and-Cut procedure. □

The learned model cannot itself certify a node. In particular, a high integrality probability, a low predicted cost-to-go, or a high rollout score never triggers pruning. These outputs only determine which admissible action is evaluated next.

A.7 Reproducibility Details

The training data contains 1,000 generated Set Cover instances with density 0.05, split into 800 training, 100 validation, and 100 test instances. The reported medium benchmark contains 10 instances with 500 constraints and $1{,}000$ variables. The hard benchmark contains 5 instances with $1{,}000$ constraints and $2{,}000$ variables. Both benchmark sets use generator seed 1.

The medium and hard time limits are 300 s and 600 s, respectively, with a node limit of 100,000. The primary neural configuration uses $k = 3$ policy candidates, $\gamma = 0.95$, both branch directions, root attention-based cut selection, and the configured primal heuristic. A 3-step rollout ($K = 3$) is the evaluated rollout setting.

The raw benchmark records retain wall-clock time, LP time, encoding time, policy time, rollout time, tree-node count, probe LP count, completion status, incumbent objective, and cut count. Peak memory and multi-seed confidence intervals were not recorded in the supplied runs and should be added in future experiments rather than inferred from the existing logs.