BENCHMARKING SYMBOLIC RULE DISCOVERY FOR BIOLOGICAL MOTIFS

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing methods for symbolic discovery primarily focus on recovering mathematical equations from numerical data, leaving it unclear whether they can induce valid logical rules from discrete sequences. We introduce SeqRule-Bench, a benchmark designed for symbolic rule discovery in biological motifs. Unlike conventional biological sequence analysis tasks formulated to classify or score sequences, it formulates each task as rule induction from labeled positive and negative sequences, requiring outputs in a human-readable domain-specific language. SeqRule-Bench integrates real biological data with synthetic tasks featuring ground-truth rules and informative negatives, addressing the scarcity of verified rules and the difficulty of obtaining negatives. To support systematic evaluation, we propose comprehensive metrics to analyze the trade-offs between accuracy, interpretability, and cost, including a novel boundary accuracy metric. By evaluating a broad range of methods, the benchmark reveals that even when these methods achieve high accuracy on some tasks, they still exhibit a gap in boundary accuracy and incur higher costs. The benchmark will be fully open-sourced upon publication.

1 INTRODUCTION

Recovering stable regularities from data is a central goal in scientific discovery and artificial intelligence. In biological sequence analysis, these regularities often manifest as motifs that contain local patterns or positional dependencies related to biological functions (Sandelin et al., 2004; Hulo et al., 2006). Because DNA and protein sequences are represented as strings over finite alphabets, their functional patterns can be expressed as interpretable rules using sequence symbols. These rules provide explicit hypotheses about sequence-function relationships and facilitate downstream biological validation through wet-lab experiments.

However, this advantage is often surrendered in existing biological sequence analysis. First, many motifs are represented by statistical summaries, such as scoring matrices or position weight matrices. Second, most existing methods (Hulo et al., 2006; Zhou et al., 2024; Feng et al., 2025) act as black-box predictors that do not recover human-readable rules to explain the observed outcomes. On the other hand, rule discovery is useful only when the recovered rules are accurate and generalizable.

This problem is challenging because rule discovery has long suffered from limited scalability, and many studies have revealed the limitations of advanced AI, including LLMs, in inductive reasoning (Hua et al., 2025; Li et al., 2025). At the same time, recent progress in LLM-based neuro-symbolic systems, LLM agents, and advanced reasoning models suggests that optimized LLM techniques may serve as useful tools for symbolic rule discovery, but most existing studies still focus on the discovery of closed-form equations (Grayeli et al., 2024; Shojaee et al., 2025), which are insufficient for representing regularities in biological sequences. Furthermore, whether LLM-based methods can recover logical rules from data remains poorly understood. Above all, the current situation indicates *a lack of a unified and standardized benchmark to evaluate the discovery of symbolic rules*.

To address this gap, we introduce *SeqRule-Bench*, a benchmark for symbolic rule discovery from sequence data. The central question we study is whether advanced LLM-based methods can move beyond fitting statistical regularities and induce more essential, human-interpretable rules from labeled sequences. As shown in Figure 1, each task in *SeqRule-Bench* is formulated as rule induction from labeled positive and negative examples. Methods are required to output domain-specific

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.

Table 1: Overview of major sequence datasets used in *SeqRule-Bench*. In the Task column, the abbreviation after the semicolon denotes the task alias.

Task	Label Semantics	Source	Rule Available
Protein: PROSITE signature rule recognition; PS	*Positive:* sequences with the target motif, domain, or site; *negative:* background sequences without the corresponding feature.	Real proteins; database annotations.	Partial
DNA: JASPAR TFBS recognition; MA	*Positive:* transcription factor binding sites; *negative:* background DNA fragments.	Real DNA; experimental labels.	No
DNA: Functional site/regulatory sequence recognition, including 4mC, 5mC, 6mA, and promoter; Gene	*Positive:* methylation-site or promoter windows; *negative:* non-target sites or non-promoter sequences.	Real DNA; experimental, annotation, or benchmark labels.	No
Protein: Functional peptide sequence recognition, including signal peptide and antimicrobial peptide (AMP); AMP	*Positive:* signal peptides, AMPs, or highly active peptides; *negative:* non-signal peptides, non-AMPs, or low-activity peptides.	Real peptides; experimental, database, or annotation labels.	No
Synthetic: Symbolic rule recognition with DSL tasks; V1–V5	*Positive:* sequences satisfying predefined DSL rules; *negative:* sequences violating the rules.	Virtual sequences; rule-assigned labels.	Yes

symbolic rules in a predefined Domain-Specific Language (DSL). We further include traditional symbolic methods and black-box predictive models as reference baselines. The evaluation protocol measures predictive performance, rule complexity, and execution cost. However, standard metrics are insufficient for evaluating rule discovery, as a discovered rule may achieve high accuracy by exploiting superficial shortcuts rather than recovering the decision boundary. To address this, we propose a boundary accuracy metric to distinguish between these two cases. Given a ground-truth (GT) rule, this metric constructs minimally perturbed sequences that violate the GT logic but retain most statistical feature and evaluates whether the learned rule changes its prediction accordingly.

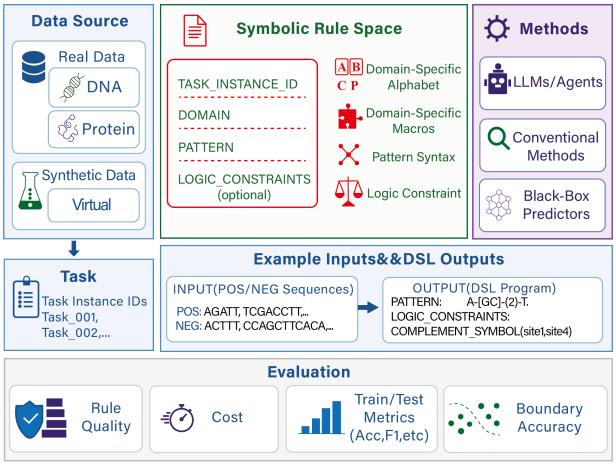


Figure 1: Main components of *SeqRule-Bench*.

Discovering essential logical regularities implies both summarizing rules from statistical patterns and recovering known underlying rules when they exist. *SeqRule-Bench* is thus designed to evaluate both abilities by combining real biological sequence tasks and synthetic symbolic sequence tasks, where synthetic datasets complement real data by providing controlled tasks with known ground-truth rules and abundant negatives, enabling direct evaluation of rule recovery across diverse rule structures.

For real data, *SeqRule-Bench* collects biological sequence tasks from established community resources, including JASPAR (Sandelin et al., 2004; Baydar et al., 2026) for transcription-factor binding sites and PROSITE (Hulo et al., 2006; Sigrist et al., 2026) for protein sequence signatures. We also include functional sequence classification tasks such as signal-peptide recognition (Armenteros et al., 2019; Teufel et al., 2022), etc, covering a spectrum from canonical motifs to functional site recognition that may include motifs. For synthetic data, we prompt an LLM to generate valid DSL rules and then construct examples from these rules. Negative examples are produced by rejection sampling, with additional hard negatives that partially match the rule but violate the pattern or logic constraints. The main tasks are summarized in Table 1.

Our main contributions are as follows:

- We introduce *SeqRule-Bench*, a benchmark for symbolic rule discovery in biological motifs. It features a biologically motivated DSL for human-readable rules and provides comprehensive metrics, including a novel boundary accuracy score, to evaluate trade-offs between accuracy, interpretability, and cost.
- We construct a task suite that includes both real biological sequences and synthetic symbolic sequences, where real datasets evaluate rule discovery under realistic data distributions, and synthetic tasks featuring ground-truth rules and informative negative.
- We evaluate a broad spectrum of methods, from advanced LLM-based approaches and traditional symbolic methods to black-box predictive models.

The experimental results reveal clear differences across methods. Overall, existing methods show promise in predictive accuracy, but substantial room remains for improving decision boundary recovery and reducing computational cost.

2 Related Work

Induction of patterns from positive and negative samples. Learning symbolic rules from positive and negative examples has been studied in grammatical inference, regular expression inference, and inductive logic programming. 1) *Grammatical inference (GI) methods*, such as RPNI (Oncina & García, 1992), EDSM (Lang et al., 1998), and FlexFringe (Verwer & Hammerschmidt, 2017), infer formal language models from labeled strings. Their outputs are typically finite automata that are executable, but less direct in expressing compact domain-specific constraints. 2) Although *regular expression inference (REI)* (Valizadeh et al., 2024) is a special case of grammatical inference, its output is limited. More critically, for our application, many REI methods rely on overlapping string data between training and testing sets. This assumption violates the evaluation principle for biological tasks, where such overlap means data leakage. 3) *Inductive logic programming (ILP)* (Evans & Grefenstette, 2018; Cropper, 2022; Cropper et al., 2020) is a classical paradigm for learning symbolic rules from positive and negative examples. However, ILP methods and benchmarks are usually defined over relational domains and first-order logic, and are rarely tailored to raw sequence data.

Symbolic regression. Symbolic regression is related to our work at the algorithmic level because both settings search over a symbolic hypothesis space and evaluate candidate hypotheses against observed examples. Classical symbolic regression (Koza, 1992) relies on genetic programming to evolve symbolic expressions through mutation, crossover, and selection. Recent methods, such as Deep Symbolic Regression (Landajuela et al., 2021) and LLM-guided symbolic search (Shojaee et al., 2025; Grayeli et al., 2024), further improve the efficiency of exploring large symbolic spaces. At the benchmark level, SRBench (Cava et al., 2021) and LLM-SRBench (Cava et al., 2021) provide standardized evaluation settings for symbolic regression methods, including recent LLM-based approaches. Despite this methodological overlap, the target problem is different because they focus on recovering mathematical expressions from numerical data, while we aim at discovering logical rules from discrete symbolic sequences.

LLM-based rule-induction. LLMs are promising tools for symbolic rule induction due to their ability to generate programs and hypotheses. However, recent benchmarks indicate that induction remains a significant challenge for LLMs. MIRAGE (Li et al., 2025), InductionBench (Hua et al., 2025), and SIRBench-V1 (Hua et al., 2025) report limited reliability in inferring general rules from examples, particularly when models must infer novel patterns rather than merely applying predefined ones. Recent neuro-symbolic systems address this weakness by integrating LLMs into program search loops. Frameworks such as Hypothesis Search (Wang et al., 2024), Hypothesis Refinement (Qiu et al., 2024), LILO (Grand et al., 2024) and FunSearch (Romera-Paredes et al., 2024) employ LLMs to propose hypotheses and subsequently rely on feedback and refinement for improvement. Our work follows this path, but focuses on inducing DSL rules rather than general hypotheses.

Advanced reasoning models and LLM agents. Recent reasoning-oriented LLM, such as GPT-series and Gemini-series reasoning models, as well as LLM agents like Codex [1] and Claude Code [2], improve multi-step reasoning, coding, and tool use, and have demonstrated broad general capabilities. Their reasoning ability has not been systematically evaluated in these related areas.

[1] https://openai.com/index/introducing-gpt-5-3-codex/

[2] https://code.claude.com/docs/en/overview

Table 2: Comparison of SeqRule-Bench with other symbolic/rule-based benchmarks.

Benchmark	Task Type	Symbolic Output	Negative Examples	Scientific-Related	LLM Evaluation	Target Format
REI	REI (GEI)	✓	✓	×	✓	Regex
LLM-SRBench	SR	✓	×	×	✓	Equation
IGGP	ILP	✓	✓	✓	×	Logic Program
InductionBench	IR	×	✓	×	✓	Label
SIRBench-V1	IR	×	×	✓	✓	String
SeqRule-Bench	**IR**	✓	✓	✓	✓	**DSL**

Pattern Syntax
- `x`: exactly one arbitrary symbol
- `x(k)`: exactly k arbitrary symbols
- `x(m,n)`: between m and n arbitrary symbols
- Repetition syntax: `<NUMBER> ::= [0-9]+`
- Allowed set `[...]` using only domain symbols
- Excluded set `{...}` using only domain symbols
- Use – between tokens
- Single valid symbol from the domain alphabet
- Left anchor < and right anchor >

Logic Constraints
- `SAME_SYMBOL(posi, posj)`
- `DIFFERENT_SYMBOL(posi, posj)`
- `COMPLEMENT_SYMBOL(posi, posj)`
- `IF(posi, A) THEN(posj, B)`
- `IF(posi, A) THEN NOT(posj, B)`
- `MIN_SYMBOL_COUNT(A, n)`
- `MAX_SYMBOL_COUNT(A, n)`
- `AND(<constraint>, <constraint>)`
- `OR(<constraint>, <constraint>)`

Figure 2: DSL for sequence rule representation.

The comparison between *SeqRule-Bench* and other related benchmarks is shown in Table 2. Overall, *SeqRule-Bench* extends the classical formulation of rule discovery to the domain of scientific sequence data. It further complements existing benchmarks by evaluating advanced baselines and requiring human-readable symbolic rules as output.

3 SeqRule-Bench

3.1 Grammar of Domain-Specific Language

Domain-specific language (Mernik et al., 2005) can be viewed as a restricted subset of general programs; DSL has more limited expressiveness but can incorporate domain-specific semantics into the language design. In this work, we design a rule grammar for biological motifs. This grammar is designed to avoid soft rules, e.g., symbol frequency, to distinguish symbolic logic from statistical features. Specifically, for a domain d, $\mathcal{A}_d$ denotes the set of valid symbols, and $\mathcal{M}_d$ denotes a set of domain-specific macros that use to interpret rules in that domain.

In DSL, we adopt an extended PROSITE-style grammar [3] to represent rules. This grammar is able to characterize conserved sequence patterns and global constraints while maintaining interpretability. Each rule consists of a core pattern and several optional logic constraints, formally defined as:

$$\mathcal{R} = (e, \Lambda), \tag{1}$$

where e describes the local structure that must appear in a sequence and Λ specifies global logic constraints. As shown in Figure 2, the pattern component defines the syntactic primitives, while the logic constraints specify positional relationships and global requirements. The inclusion of positional relationships is motivated by the use of position matrices in the JASPAR and PROSITE profiles to characterize the motifs. To maintain readability, we do not introduce nested logic.

Domain-specific alphabet. *SeqRule-Bench* defines alphabets for both biological and synthetic sequence domains. For biological sequences, we use protein sequences defined over 20 natural amino acids $\{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$, DNA sequences over the nucleotide alphabet $\{A, T, G, C\}$. To further evaluate rule discovery beyond fixed biological alphabets, we introduce five synthetic symbolic domains with controlled alphabet sizes, ranging from a full 26-symbol alphabet $V1 : \{A, B, \ldots, Z\}$ to smaller alphabets $V2 : \{A, B, \ldots, T\}$, $V3 : \{A, B, \ldots, O\}$, $V4 : \{A, B, \ldots, J\}$, and $V5 : \{A, B, \ldots, E\}$.

[3] `https://prosite.expasy.org/prosuser.html`

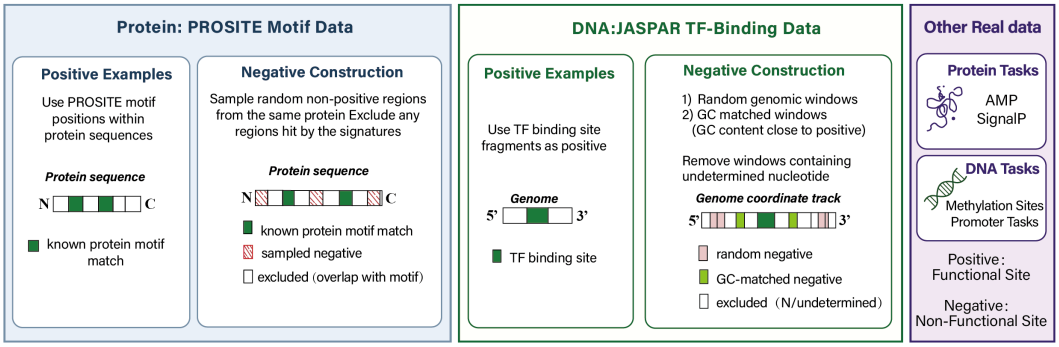


Figure 3: Positive/negative example construction for real datasets.

Domain-specific macros. *SeqRule-Bench* associates each domain with a set of macros $\mathcal{M}_d$ derived from its alphabet configuration. Unlike ordinary grammar tokens, these macros do not have to appear literally in the rules. Instead, they provide domain-specific background knowledge for prompting LLMs to generate interpretable rules and implicit patterns. Specifically, $\mathcal{M}_d = (\mathcal{P}_d, \mathcal{C}_d)$, where $\mathcal{P}_d$ defines higher-level symbol groups such as hydrophobic amino acids or purine nucleotides, and $\mathcal{C}_d$ specifies valid paired-symbol relations for constraints such as `COMPLEMENT_SYMBOL(posi,posj)`.

3.2 Real-World Data

Real-world data in *SeqRule-Bench* include curated motif databases, i.e., JASPAR and PROSITE, together with several real biological sequence function-recognition tasks that may involve motifs (Figure 3). Task instances over real-world data are characterized by non-synthetic sequences with reliable annotated labels.

JASPAR TF-binding tasks. We constructed positives from JASPAR transcription-factor binding sites with genomic coordinates. The coordinates of the site were parsed into genomic intervals, and the corresponding fragments were used as positive sequences. All positive intervals, together with organism-specific blacklist intervals when available, were treated as forbidden regions. Candidate windows that overlap prohibited regions or previously sampled negatives were excluded, and sequences containing symbols other than A/C/G/T were discarded.

Negatives were sampled from the corresponding reference genome with the same length distribution as positives. For each task, we generated random genomic negatives and GC-matched negatives. For GC matching, a candidate negative x^- was accepted for a positive sequence x^+ iff:

$$|\mathrm{GC}(x^-) - \mathrm{GC}(x^+)| \le \tau, \quad \mathrm{GC}(x) = \frac{\#G(x) + \#C(x)}{|x|}. \tag{2}$$

Finally, sequences were deduplicated after reverse-complement canonicalization.

PROSITE motif tasks. The data source consists of PROSITE entries and their corresponding Swiss-Prot [4] protein sequences. We used the annotated *true positive* motif positions in PROSITE pattern or profile entries as positive examples. These fragments represent biologically significant regions that also contain the corresponding signature-rule features. Following (Necci et al., 2021) for the negative construction, we first marked all all annotated positive motif coordinates and additional PROSITE signature hits in the same protein as the excluded regions. We then sampled random regions from the remaining parts of the same proteins, using the same length as the corresponding positive motif segment. Finally, all examples were de-duplicated.

Other real-sequence tasks. 1) DNA tasks: we use genomic benchmark datasets (Feng et al., 2025) covering methylation-site recognition and promoter recognition. 2) Protein tasks: we include functional-sequence recognition datasets, such as signal-peptide (Armenteros et al., 2019; Teufel et al., 2022) and antimicrobial-peptide classification (Szymczak et al., 2023; Pandi et al., 2023). We retain the original positive and negative labels provided by these datasets and apply sequence-level deduplication before constructing the final task instances.

[4] `https://www.uniprot.org/uniprotkb?query=reviewed:true`

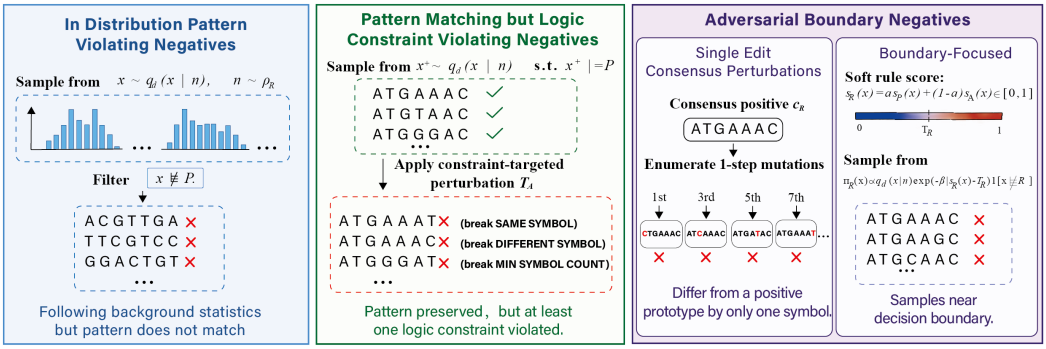


Figure 4: Negative example composition for synthetic data.

3.3 Synthetic Data

Synthetic data are constructed through a two-stage pipeline: an LLM first synthesizes symbolic rules, after which sequences are sampled from a reference distribution and labeled according to rule satisfaction. During rule generation, LLM is provided with the domain-specific alphabet, macros, representative rule examples, and additional biological prior knowledge; then it is prompted to synthesize valid symbolic rules under predefined complexity and length requirements (Appendix A.8.1). In *SeqRule-Bench*, we generate rule variants consisting of patterns only, as well as rules that combine patterns with varying numbers of logic constraints, to comprehensively evaluate method performance.

When sampling data, the main difficulty lies in constructing informative negatives. To address this issue, we synthesize three additional levels of negative samples and combine them with randomly sampled background negatives to form the complete negative set (Figure 4). All synthesized negatives satisfy: $x \not\models R$, i.e., they violate the pattern P or at least one logical constraint in Λ.

Negatives affected by the distribution pattern. The random negatives are sampled from a reference distribution ρ_R:

$$x \sim q_d(x \mid n), \qquad n \sim \rho_R, \tag{3}$$

and retain only candidates that violate the core pattern: $x \not\models P$. Here q_d can be estimated from empirical symbol frequencies, position frequency matrices, or large domain corpora.

Negatives based on pattern-matching but logic-violating. In this case, we first sample a candidate that satisfies the pattern:

$$x^+ \sim q_d(x \mid n) \quad \text{s.t.} \quad x^+ \models P, \tag{4}$$

and then apply a constraint-targeted perturbation operator T_Λ to obtain $\tilde{x} = T_\Lambda(x^+)$, such that:

$$\tilde{x} \models P \quad \text{and} \quad \exists \lambda \in \Lambda \ \text{ s.t. } \ \lambda(\tilde{x}) = 0. \tag{5}$$

In practice, T_Λ performs local edits that break specific logical relations, such as symbol agreement, counting constraints, conditional implications, or complementarity, while keeping the local pattern intact. These negatives are therefore highly similar to positives at the motif level and differ mainly in higher-order structure. When $\Lambda = \varnothing$, this tier is omitted.

Adversarial boundary negatives. Finally, we generate near-boundary negatives that are minimally different from valid positives, but still violate the rule. We use two complementary constructions:

First, we perform *single-edit consensus perturbation*. Let c_R denote a consensus or canonical positive sequence induced by rule R. The enumerated one-step mutation is formulated as:

$$\tilde{x}^{(i \to a)} = c_R[1{:}i{-}1] \oplus a \oplus c_R[i{+}1{:}n], \qquad a \in \mathcal{A}_d \setminus \{c_R[i]\}, \tag{6}$$

and retain those for which $\tilde{x}^{(i \to a)} \not\models R$. This produces negatives that differ from a positive prototype by only a single symbol.

Second, we performed *boundary-focused MCMC sampling*. The soft rule score is defined as:

$$s_R(x) = \alpha\, s_P(x) + (1 - \alpha)\, s_\Lambda(x), \qquad s_R(x) \in [0, 1], \tag{7}$$

where $s_P(x)$ measures pattern compatibility and $s_\Lambda(x)$ measures logical satisfaction. The data distribution is as follows:

$$\pi_R(x) \propto q_d(x \mid n)\, \exp\big(-\beta|s_R(x) - \tau_R|\big)\, \mathbf{1}[x \not\models R], \tag{8}$$

where τ_R is the rule threshold and $\beta > 0$ controls concentration near the decision boundary. This procedure yields hard negatives that are close to valid positives under a continuous relaxation of the rule and is particularly useful for stress-testing boundary sensitivity.

3.4 Evaluation Metrics

The evaluation mainly analyzes the quality and generalization of the discovered rules, as well as the cost of the discovery process. The major metrics are briefly introduced below (the detailed descriptions are show in Appendix A.2):

Boundary accuracy evaluates whether a discovered rule preserves the decision-boundary behavior of the ground-truth rule. We construct a set of minimally perturbed sequence pairs $\mathcal{P}_\mathcal{G}$, where each pair (x, x') satisfies $\mathcal{G}(x) \neq \mathcal{G}(x')$. These pairs are generated by perturbing pattern-level or logic-level constraints, independently of the learned rules being evaluated, and are shared across all methods. Boundary accuracy measures the proportion of these ground-truth boundary-crossing pairs for which the learned rule correctly classifies both endpoints. Concretely, given the set $\mathcal{P}_\mathcal{G}$ of ground-truth boundary-crossing pairs and a learned rule $\mathcal{R}$, these metrics are formulated as:

$$\mathrm{BoundaryAcc} = \frac{1}{|\mathcal{P}_\mathcal{G}|} \sum_{(x,x')\in\mathcal{P}_\mathcal{G}} \mathbb{1}[\mathcal{R}(x) = \mathcal{G}(x) \wedge \mathcal{R}(x') = \mathcal{G}(x')]\,, \qquad \mathrm{Coverage} = \frac{|\mathcal{B}_{\mathrm{covered}}|}{|\mathcal{B}_\mathcal{G}|}, \tag{9}$$

where $\mathcal{B}_\mathcal{G}$ enumerates the boundary-relevant components of the ground-truth rule and $\mathcal{B}_{\mathrm{covered}}$ is the subset that actually is exercised by the boundary-crossing pairs generated. $\mathrm{BoundaryAcc}$ is read against the GT rule rather than stored labels, so by construction it is immune to noise from the test-set label and to the class imbalance ratio. The detailed definitions and motivation are given in Appendix A.2.1, and the full empirical study that validates correctness, estimator stability, robustness to corruption and label noise, and discriminative power against Acc/F1 is reported in Appendix A.3.

Predictive performance measures the classification ability and generalization of the generated rule. We report accuracy on a held-out balanced test set.

Cost: For LLM-based baselines, token usage incurred during rule generation is reported.

4 Experiments

We conducted all experiments on a server equipped with eight NVIDIA A800-SXM4-80GB GPUs and an Intel Xeon Platinum 8358 CPU (2.60 GHz, 64 cores, 128 threads). The baselines follow their default settings. We evaluate few-shot LLM, agentic methods (i.e. Codex with GPT-5.4 and Claude Code with Sonnet 4.6), advanced LLM reasoning models (i.e., GPT-5.4[5], Claude Sonnet 4.6[6], and Gemini-3.1-Pro-Preview[7]), LLM-SR (Shojaee et al., 2025) and LLM-based rule induction methods (i.e. Self-Refinement (Madaan et al., 2023) and Hypothesis Refinement (Qiu et al., 2024)). Except for the agentic and few-shot LLM methods, all other LLM-based methods adopt GPT-5.4 as the default backbone model. For comparison, black-box predictors (i.e., ESM-2 (Lin et al., 2023), DNABERT-2 (Zhou et al., 2024), 1DCNN, and 1D ResNet) and conventional methods (i.e., FlexFringe (Verwer & Hammerschmidt, 2017) and symbolic search) are also included. The prompt template is provided in Appendix A.8.2, while the concrete settings are detailed in Appendix A.7. The test set is evaluated on a balanced subset, with 500 positive and 500 negative examples by default. Our code is available at `https://anonymous.4open.science/r/seqrule-anon-4d86107d-55B3/`.

[5] `https://openai.com/index/introducing-gpt-5-4/`

[6] `https://www.anthropic.com/news/claude-sonnet-4-6`

[7] `https://ai.google.dev/gemini-api/docs/models/gemini-3.1-pro-preview`

Table 3: Boundary accuracy on real and synthetic subsets. Intervals below the values are bootstrap 95% confidence intervals over the per-rule `BoundaryAcc` (1,000 resamples), the mean `Coverage` value across subsets ranges from 0.80 to 1.00. Synthetic subsets are denoted Vk-{s,m,c}, where Vk is the virtual domain with alphabet size $|\Sigma| = 26, 15, 5$ for $k = 1, 3, 5$, and s/m/c indicate the complexity tier (s = simple, 0 logic constraints; m = medium, 2–3; c = complex, 4–5).

Method	Real subset	Synthetic subsets								
	PS-S	V1-s	V1-m	V1-c	V3-s	V3-m	V3-c	V5-s	V5-m	V5-c
Claude Code	**0.64** [0.55, 0.74]	**0.37** [0.29, 0.48]	**0.68** [0.31, 0.97]	**0.54** [0.39, 0.77]	**0.83** [0.75, 0.93]	**0.66** [0.47, 0.87]	**0.39** [0.14, 0.63]	0.61 [0.39, 0.84]	**0.70** [0.57, 0.84]	**0.59** [0.37, 0.79]
Codex	0.53 [0.43, 0.64]	0.36 [0.22, 0.54]	0.62 [0.40, 0.86]	0.48 [0.31, 0.76]	0.48 [0.24, 0.68]	0.59 [0.29, 0.88]	0.39 [0.32, 0.46]	**0.75** [0.55, 0.95]	0.67 [0.47, 0.85]	0.51 [0.27, 0.76]
Few-Shot/claude	0.28 [0.19, 0.38]	0.16 [0.07, 0.27]	0.54 [0.25, 0.84]	0.32 [0.11, 0.66]	0.38 [0.22, 0.53]	0.13 [0.08, 0.19]	0.17 [0.06, 0.27]	0.28 [0.15, 0.45]	0.32 [0.21, 0.43]	0.15 [0.03, 0.27]
Few-Shot/gemini	0.41 [0.28, 0.54]	0.03 [0.01, 0.06]	0.51 [0.19, 0.82]	0.35 [0.07, 0.69]	0.40 [0.12, 0.69]	0.31 [0.11, 0.57]	0.14 [0.06, 0.24]	0.40 [0.26, 0.60]	0.16 [0.04, 0.33]	0.33 [0.09, 0.57]
Few-Shot/gpt	0.09 [0.02, 0.17]	0.07 [0.02, 0.12]	0.31 [0.14, 0.46]	0.15 [0.05, 0.26]	0.35 [0.15, 0.54]	0.14 [0.01, 0.28]	0.07 [0.03, 0.13]	0.19 [0.08, 0.30]	0.21 [0.06, 0.40]	0.08 [0.04, 0.14]
Hypothesis Refinement	0.51 [0.42, 0.59]	0.18 [0.10, 0.27]	0.37 [0.17, 0.67]	0.28 [0.19, 0.35]	0.53 [0.31, 0.77]	0.19 [0.10, 0.28]	0.19 [0.10, 0.29]	0.39 [0.28, 0.54]	0.16 [0.06, 0.25]	0.29 [0.13, 0.52]
Self-Refinement	0.35 [0.23, 0.46]	0.12 [0.05, 0.23]	0.44 [0.16, 0.73]	0.37 [0.33, 0.43]	0.45 [0.21, 0.65]	0.24 [0.15, 0.34]	0.11 [0.08, 0.14]	0.27 [0.14, 0.42]	0.23 [0.08, 0.38]	0.33 [0.13, 0.57]
LLM-SR	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]	0.06 [0.03, 0.08]	0.07 [0.01, 0.12]	0.00 [0.00, 0.00]	0.14 [0.00, 0.28]	0.24 [0.01, 0.48]	0.01 [0.00, 0.02]
FlexFringe	0.14 [0.11, 0.16]	0.08 [0.07, 0.10]	0.24 [0.12, 0.39]	0.17 [0.13, 0.22]	0.16 [0.11, 0.21]	0.18 [0.12, 0.22]	0.10 [0.08, 0.12]	0.43 [0.32, 0.55]	0.52 [0.41, 0.63]	0.39 [0.24, 0.56]
GA	0.18 [0.13, 0.23]	0.05 [0.02, 0.10]	0.26 [0.08, 0.47]	0.16 [0.08, 0.26]	0.23 [0.04, 0.45]	0.23 [0.07, 0.41]	0.10 [0.03, 0.21]	0.30 [0.12, 0.59]	0.17 [0.07, 0.28]	0.13 [0.03, 0.29]
SA	0.06 [0.04, 0.07]	0.03 [0.01, 0.05]	0.02 [0.00, 0.07]	0.11 [0.05, 0.18]	0.02 [0.00, 0.04]	0.07 [0.03, 0.11]	0.07 [0.02, 0.13]	0.13 [0.06, 0.24]	0.10 [0.04, 0.19]	0.08 [0.03, 0.17]

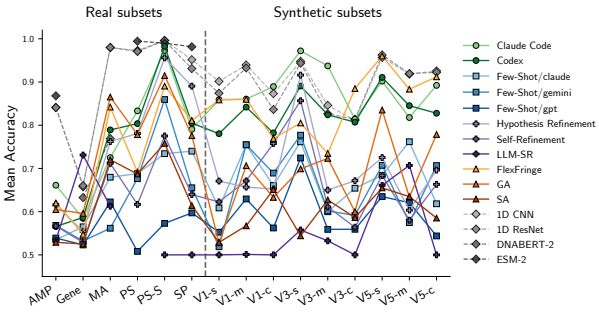


Figure 5: Mean test accuracy across the 15 benchmark subsets (6 real + 9 synthetic). Solid lines are rule-discovery baselines, coloured by family (LLM-Based, symbolic search, agentic LLM); dashed lines are black-box predictors ($\tau = 0.05$).

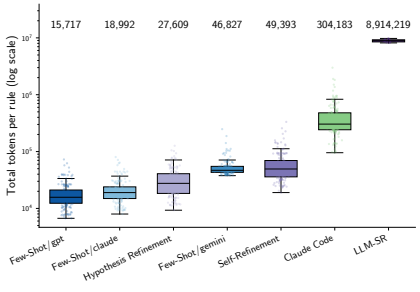


Figure 6: Per-rule total token consumption for the LLM baselines that record full token usage. The median is annotated above each box.

4.1 Trade-offs between Performance and Cost

In this experiment, for the Few-Shot LLM, Hypothesis Refinement, and Self-Refinement, 800 randomly balanced positive and negative samples were loaded into the context, while all other methods utilized the full dataset. Table 3 and Figure 5 show that Claude Code achieves substantially higher BoundaryAcc and accuracy than the other rule-discovery methods, with its best performance around 0.80 BoundaryAcc and more than 0.90 test accuracy and the improvement is concentrated on only part of the synthetic subsets. Compared with their corresponding base LLMs, Claude Code and Codex yield clear accuracy gains, suggesting that harnessing and stronger agentic reasoning are important sources of improvement. Under the same base LLM, Hypothesis Refinement and Self-Refinement both improve over Few-Shot LLMs. Hypothesis Refinement further outperforms Self-Refinement in many cases of accuracy, indicating that generating multiple candidate rules and selecting the best one is an effective strategy when labeled examples are available, but this advantage is not stable in terms of BoundaryAcc. In several real subsets, GA achieves similar or slightly higher accuracy than FlexFringe, indicating that the proposed hard-logic grammar is sufficiently expressive to represent meaningful sequence rules.

On datasets with ground-truth rules, agentic methods can approach black-box predictive accuracy, while on real datasets without explicit rules, they remain comparable to traditional symbolic methods. In contrast, the common LLM-based strategy LLM-SR used in many scientific discovery settings performs poorly in this benchmark.

Figure 6 shows three distinct token-cost regimes. Non-agentic LLM baselines consume 10^4–10^5 tokens per rule, with median costs ranging from 1.6×10^4 for Few-Shot/gpt to 4.9×10^4 for Self-Refinement. Their tight distributions suggest that cost is mainly determined by fixed few-shot examples and short refinement histories. Self-Refinement and Few-Shot/gemini are slightly more expensive, due to accumulated feedback context and longer reasoning outputs, respectively. Claude Code is more costly, with a median of 3.0×10^5 tokens per rule and a tail exceeding 10^6. This is about one order of magnitude higher than non-agentic baselines, mainly because the agent reads the full training data, executes analysis scripts, and inspects intermediate outputs. LLM-SR sits in a third regime, with a median of 8.9×10^6 tokens per rule. The cost reflects LLM-SR's iterative program-search loop, in which the LLM proposes thousands of candidate rule skeletons that are externally re-scored. Combined with its weak accuracy on most subsets in Figure 5, the cost–quality trade-off of LLM-SR is dominated by every other LLM-based method evaluated here.

4.2 Scalability Analysis

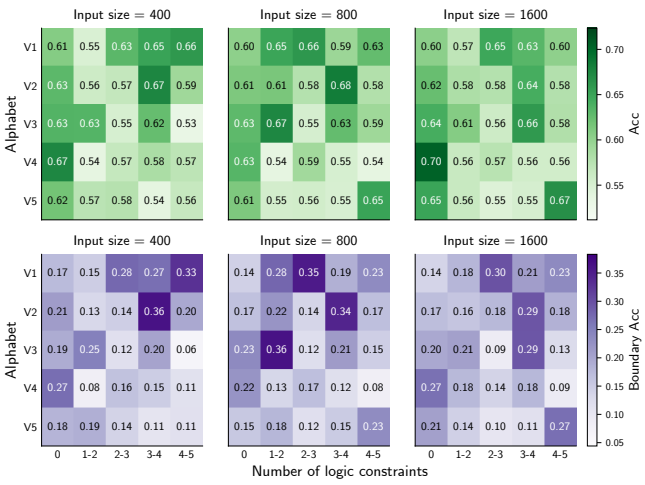


Figure 7: Few-Shot/GPT-5.4 performance vs. input data size. Each heatmap is 5×5: the vertical axis varies the alphabet V_1–V_5 with alphabet sizes $|\Sigma| = 26, 20, 15, 10, 5$; the horizontal axis varies the number of logic constraints per rule. Each cell is the mean over 10 rules (two decimals), the mean `Coverage` value across subsets ranges from 0.80 to 1.00.

From Figure 7, baselines show certain capability for rule recovery. However, their overall boundary accuracy remains low. Many results achieve relatively high accuracy, such as exceeding 0.65, yet their boundary accuracy remains capped below 0.4. Furthermore, although high accuracy is correlated with better boundary accuracy to some extent, they are not strictly proportional. This gap and inconsistency suggests that the model may still rely on statistical regularities or shortcut features to approximate the target rule, rather than truly recovering the decision boundaries.

Finally, increasing the number of input examples brings only a modest improvement in accuracy. This suggests that simply providing more labeled examples is insufficient to ensure more faithful rule recovery.

5 Conclusion

In this paper, we introduced *the SeqRule-Bench*, a benchmark that reformulates biological sequence analysis as a symbolic rule discovery from labeled positive and negative sequences. *SeqRule-Bench* combines real biological tasks with synthetic tasks equipped with ground-truth rules and informative negatives. In addition, it provides a unified evaluation protocol covering accuracy, computational cost, and a boundary accuracy metric. Experimental results show that high accuracy does not guarantee true rule recovery, concretely: 1) Agentic LLMs can achieve near-black-box accuracy on tasks derived from ground-truth rules; 2) Refinement further improves performance over standard few-shot LLMs, indicating the value of exploring self-evolution; 3) Increasing the number of candidates can improve accuracy, but does not necessarily lead to learning the true underlying rules; 4) Higher costs do not guarantee improved performance, especially in terms of boundary accuracy.

AI USE STATEMENT

LLMs are used as core components in several rule-discovery baselines and in the synthetic rule-generation pipeline. We describe the evaluated LLM-based methods, model settings, prompt templates, and baseline-specific prompt adaptations in the experimental configuration and prompt-template appendices.

We used generative AI tools to assist with code debugging, literature searches, and language editing. The authors have manually reviewed all AI-assisted content included in this work.

ETHICS STATEMENT

The work uses public biological sequence resources, existing benchmark datasets, synthetic data, and LLM/API baselines; it does not involve human subjects or private personal data, so this work complies with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

The paper describes the data sources, synthetic generation pipeline, train/test construction, baseline configurations, evaluation metrics, and compute setup. Appendix A.7, Appendix A.8.2, and Appendix A.3 provide the detailed configurations and protocols needed to reproduce the reported results.

REFERENCES

José Juan Almagro Armenteros, Konstantinos D. Tsirigos, Casper Kaae Sønderby, Thomas Nordahl Petersen, Ole Winther, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. Signalp 5.0 improves signal peptide predictions using deep neural networks. *Nature Biotechnology*, 37: 420–423, 2019. doi: 10.1038/s41587-019-0036-z.

Damla Ovek Baydar, Ieva Rauluseviciute, Dina R. Aronsen, Romain Blanc-Mathieu, Ine Bonthuis, Herman de Beukelaer, Katalin Ferenc, Alice Jegou, Vipin Kumar, Roza Berhanu Lemma, Jérémy Lucas, Mathis Pochon, Chang M. Yun, Vivekanandan Ramalingam, Salil Sanjay Deshpande, Aman Patel, Georgi K. Marinov, Austin T. Wang, Alejandro Aguirre, Jaime A. Castro-Mondragon, Damir Baranasic, Jeanne Chèneby, Sveinung Gundersen, Morten Johansen, Aziz Khan, Marieke L. Kuijjer, Eivind Hovig, Boris Lenhard, Albin Sandelin, Klaas Vandepoele, Wyeth W. Wasserman, François Parcy, Anshul Kundaje, and Anthony Mathelier. Jaspar 2026: Expansion of transcription factor binding profiles and integration of deep learning models. *Nucleic Acids Research*, 54(D1): D184–D193, 2026. doi: 10.1093/nar/gkaf1209.

William La Cava, Bogdan Burlacu, Marco Virgolin, Michael Kommenda, Patryk Orzechowski, Fabrício Olivetti de França, Ying Jin, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. *Advances in neural information processing systems*, 2021 DB1:1–16, 2021.

Andrew Cropper. Learning logic programs though divide, constrain, and conquer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 6446–6453, 2022. doi: 10.1609/aaai.v36i6.20596.

Andrew Cropper, Richard Evans, and Mark Law. Inductive general game playing. *Machine Learning*, 109:1393–1434, 2020. doi: 10.1007/s10994-019-05843-w.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Haonan Feng, Lang Wu, Bingxin Zhao, Chad Huff, Jianjun Zhang, Jia Wu, Lifeng Lin, Peng Wei, and Chong Wu. Benchmarking dna foundation models for genomic and genetic tasks. *Nature Communications*, 16:10780, 2025. doi: 10.1038/s41467-025-65823-8.

Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=TqYbAWKMIe.

Arya Grayeli, Atharva Sehgal, Omar Costilla Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=B7S4jJGlvl.

Thomas Hayes, Roshan Rao, Halil Akin, Nicholas J. Sofroniew, Deniz Oktay, Zeming Lin, Robert Verkuil, Vincent Q. Tran, Jonathan Deaton, Marius Wiggert, Rohil Badkundri, Irhum Shafkat, Jun Gong, Alexander Derry, Raul S. Molina, Neil Thomas, Yousuf Khan, Chetan Mishra, Carolyn Kim, Liam J. Bartie, Matthew Nemeth, Patrick D. Hsu, Tom Sercu, Salvatore Candido, and Alexander Rives. Simulating 500 million years of evolution with a language model. *Science*, 387:850–858, 2025. doi: 10.1126/science.ads0018.

Wenyue Hua, Tyler Wong, Fei Sun, Liangming Pan, Adam Jardine, and William Yang Wang. Inductionbench: Llms fail in the simplest complexity class. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26526–26546, 2025. doi: 10.18653/v1/2025.acl-long.1287.

Nicolas Hulo, Amos Bairoch, Virginie Bulliard, Lorenzo Cerutti, Edouard De Castro, Petra S. Langendijk-Genevaux, Marco Pagni, and Christian J. A. Sigrist. The prosite database. *Nucleic Acids Research*, 34(suppl_1):D227–D230, 2006. doi: 10.1093/nar/gkj063.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 9780262111706.

Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, volume 139, pp. 5979–5989. PMLR, 2021. URL https://proceedings.mlr.press/v139/landajuela21a.html.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of the 4th International Colloquium on Grammatical Inference*, ICGI '98, pp. 1–12, Berlin, Heidelberg, 1998. Springer-Verlag. ISBN 3540647767.

Jiachun Li, Pengfei Cao, Zhuoran Jin, Yubo Chen, Kang Liu, and Jun Zhao. Mirage: Evaluating and explaining inductive reasoning process in language models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tZCqSVncRf.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37:316–344, 2005. doi: 10.1145/1118890.1118892.

Marco Necci, Damiano Piovesan, CAID Predictors, DisProt Curators, and Silvio C. E. Tosatto. Critical assessment of protein intrinsic disorder prediction. *Nature Methods*, 18:472–481, 2021. doi: 10.1038/s41592-021-01117-3.

José Oncina and Pedro García. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E. Vidal (eds.), *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*, pp. 49–61. World Scientific, 1992.

Amir Pandi, David Adam, Amir Zare, Van Tuan Trinh, Stefan L. Schaefer, Marie Burt, Björn Klabunde, Elizaveta Bobkova, Manish Kushwaha, Yeganeh Foroughijabbari, Peter Braun, Christoph Spahn, Christian Preußer, Elke Pogge von Strandmann, Helge B. Bode, Heiner von Buttlar, Wilhelm Bertrams, Anna Lena Jung, Frank Abendroth, Bernd Schmeck, Gerhard Hummer, Olalla Vázquez, and Tobias J. Erb. Cell-free biosynthesis combined with deep learning accelerates de novo-development of antimicrobial peptides. *Nature Communications*, 14:7197, 2023. doi: 10.1038/s41467-023-42434-9.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=bNt7oajl2a.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Albin Sandelin, Wynand Alkema, Pär Engström, Wyeth W. Wasserman, and Boris Lenhard. Jaspar: An open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, 32(suppl_1):D91–D94, 2004. doi: 10.1093/nar/gkh012.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. Llm-sr: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m2nmp8P5in.

Christian J A Sigrist, Béatrice A Cuche, Edouard de Castro, Elisabeth Coudert, Nicole Redaschi, and Alan Bridge. The prosite database for protein families, domains, and sites. *Nucleic Acids Research*, 54(D1):D451–D458, 2026. doi: 10.1093/nar/gkaf1188.

Paulina Szymczak, Marcin Możejko, Tomasz Grzegorzek, Radosław Jurczak, Marta Bauer, Damian Neubauer, Karol Sikora, Michał Michalski, Jacek Sroka, Piotr Setny, Wojciech Kamysz, and Ewa Szczurek. Discovering highly potent antimicrobial peptides with deep generative model hydramp. *Nature Communications*, 14:1453, 2023. doi: 10.1038/s41467-023-36994-z.

Felix Teufel, José Juan Almagro Armenteros, Alexander Rosenberg Johansen, Magnús Halldór Gíslason, Silas Irby Pihl, Konstantinos D. Tsirigos, Ole Winther, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. Signalp 6.0 predicts all five types of signal peptides using protein language models. *Nature Biotechnology*, 40:1023–1025, 2022. doi: 10.1038/s41587-021-01156-3.

Mojtaba Valizadeh, Philip John Gorinski, Ignacio Iacobacci, and Martin Berger. Correct and optimal: The regular expression inference challenge. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024.

Sicco Verwer and Christian A. Hammerschmidt. Flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution*, pp. 638–642, 2017. doi: 10.1109/ICSME.2017.58.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=G7UtIGQmjm.

Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana V Davuluri, and Han Liu. DNABERT-2: Efficient foundation model and benchmark for multi-species genomes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oMLQB4EZE1.

Table 4: Key notations.

Notation	Semantics
$\mathcal{R} = (e, \Lambda)$	Learned rule, consisting of a symbolic pattern e and logical constraints Λ.
$\mathcal{G}$	Ground-truth rule.
$\mathcal{A}_d$	Set of valid symbols for domain d.
$\mathcal{M}_d$	Domain-specific macros used to interpret rules in domain d.
$\mathcal{P}_d$	Symbol-group macros in domain d, such as hydrophobic amino acids or purine nucleotides.
$\mathcal{C}_d$	Paired-symbol relation macros in domain d.
x_i, y_i	The i-th sequence and its label.
D, D^+, D^-	Evaluation dataset, positive subset, and negative subset.
$\mathcal{M}(e, x)$	Set of matches produced by pattern e on sequence x.
m	A specific match in $\mathcal{M}(e, x)$.
$\Lambda(m, x)$	Whether logical constraints are satisfied for match m on sequence x.
$\Omega(x_i)$	Valid perturbation space of sample x_i.
x'	perturbed variant of an original sequence.
$\mathcal{C}_{\mathcal{G}}(x_i)$	Set of perturb variants of x_i.
$\mathcal{P}_{\mathcal{G}}$	Set of generated ground-truth boundary-crossing boundary-crossing pairs.
$\mathrm{BoundaryAcc}$	Boundary accuracy score.
$\mathrm{Coverage}$	Coverage of ground-truth boundary components by perturbations.
$\mathrm{MeanFlipDist}$	Average perturbation distance between original and perturbed sequences.
$\mathrm{Comp}(\mathcal{R})$	Structural complexity of the learned rule.
t_j, L	The j-th pattern token and the total number of pattern tokens.
λ	A logical constraint in Λ.
$c_{\mathrm{pat}}, c_{\mathrm{logic}}$	Complexity cost functions for pattern tokens and logic constraints.
τ	GC-matching threshold used when sampling GC-matched negative DNA sequences.

A Appendix

A.1 Notation and Definitions

The key notations and their definitions are summarized in Table 4.

A.2 Additional Details and Experiments on the Evaluation Metric

A.2.1 Boundary Accuracy

Classification metrics such as Accuracy and F1 alone may be insufficient to evaluate rule discovery. A learned rule may achieve high predictive performance by exploiting superficial shortcuts, such as sequence length or GC content, without capturing the underlying symbolic mechanism. To evaluate classification correctness across local ground-truth decision boundaries, we introduce *boundary accuracy*.

Let $\mathcal{M}(e, x)$ denote the set of matches produced by expression e on sample x, and let Λ denote the associated logical constraints. A learned rule $\mathcal{R}$ is satisfied when at least one match satisfies these constraints:

$$\mathcal{R}(x) = 1 \iff \exists m \in \mathcal{M}(e, x) \text{ such that } \Lambda(m, x) = \mathrm{True}. \tag{10}$$

Ground-truth boundary-crossing pairs. Let $\mathcal{G}$ denote the ground-truth rule and D the set of source sequences. For each $x_i \in D$, let $\Omega(x_i)$ denote the candidate variants generated using valid, task-specific perturbations. These perturbations modify boundary-relevant components of the ground-truth rule, while keeping changes to the sequence small to retain the statistical feature.

We retain only variants whose ground-truth predictions differ from those of their source sequences:

$$\mathcal{C}_{\mathcal{G}}(x_i) = \{x' \in \Omega(x_i) : \mathcal{G}(x_i) \neq \mathcal{G}(x')\}. \tag{11}$$

The resulting set of GT boundary-crossing pairs is:

$$\mathcal{P}_{\mathcal{G}} = \{(x_i, x') : x_i \in D,\ x' \in \mathcal{C}_{\mathcal{G}}(x_i)\}. \tag{12}$$

Pair generation and filtering depend only on the GT rule and the prescribed perturbation procedure, not on any learned rule.

Boundary accuracy. Boundary accuracy measures the proportion of GT boundary-crossing pairs for which the learned rule correctly classifies both endpoints:

$$\mathrm{BoundaryAcc}(\mathcal{R}, \mathcal{G}, D) = \frac{1}{|\mathcal{P}_{\mathcal{G}}|} \sum_{(x,x') \in \mathcal{P}_{\mathcal{G}}} \mathbb{1}[\mathcal{R}(x) = \mathcal{G}(x) \wedge \mathcal{R}(x') = \mathcal{G}(x')]. \tag{13}$$

A pair contributes to the score only when both predictions match their respective ground-truth labels. A high boundary accuracy indicates agreement with the ground-truth rule on both sides of the sampled local boundaries. A learned rule that fails to respond correctly to boundary-crossing perturbations receives a low score, even if its accuracy on ordinary test samples is high.

Boundary coverage. Boundary coverage is designed to ensure that the evaluation set exercises a sufficiently broad range of GT rule components:

$$\mathrm{Coverage}(\mathcal{P}_{\mathcal{G}}) = \frac{|\mathcal{B}_{\mathrm{covered}}(\mathcal{P}_{\mathcal{G}})|}{|\mathcal{B}_{\mathcal{G}}|}. \tag{14}$$

Here, $\mathcal{B}_{\mathcal{G}}$ is the predefined set of boundary-relevant pattern and logic components in the GT rule, and $\mathcal{B}_{\mathrm{covered}}(\mathcal{P}_{\mathcal{G}})$ is the subset perturbed by at least one retained boundary-crossing pair. Coverage helps assess the reliability of boundary accuracy by indicating the extent of the GT decision boundary covered by the evaluation. In this paper, for constructing minimal perturbed sequence that across the boundary, we restrict the number of character edits to one or two.

A.2.2 Rule Complexity

Predictive performance alone does not indicate whether a discovered rule is concise or interpretable. A model may fit the data using an overly specific rule that memorizes local artifacts in the training examples. Therefore, we report the structural complexity of each discovered rule using operator-based costs.

For a learned rule $\mathcal{R} = (e, \Lambda)$, we define the complexity of the rule as the sum of the complexity of the pattern and the complexity of logic:

$$\mathrm{Comp}(\mathcal{R}) = \mathrm{Comp}_{\mathrm{pat}}(e) + \mathrm{Comp}_{\mathrm{logic}}(\Lambda). \tag{15}$$

The pattern complexity measures the number of structural units required to describe the local sequence pattern. Let e be parsed into a sequence of pattern tokens: $e = (t_1, t_2, \ldots, t_L)$, where each token can be a literal symbol, wildcard, character class, excluded class, fixed-length gap, or variable-length gap, so the total rule complexity is defined as:

$$\mathrm{Comp}_{\mathrm{pat}}(e) = \sum_{j=1}^{L} c_{\mathrm{pat}}(t_j), \tag{16}$$

where

$$c_{\mathrm{pat}}(t_j) = \begin{cases} 1, & \text{if } t_j \text{ is a literal symbol or wildcard,} \\ 2, & \text{if } t_j \text{ is a character class or excluded class,} \\ 2, & \text{if } t_j \text{ is a fixed-length gap,} \\ 3, & \text{if } t_j \text{ is a variable-length gap.} \end{cases} \tag{17}$$

This assigns higher cost to tokens that introduce additional degrees of freedom, such as character sets and variable-length gaps.

The logic complexity measures the additional global or cross-position constraints imposed by the rule:

$$\mathrm{Comp}_{\mathrm{logic}}(\Lambda) = \sum_{\lambda \in \Lambda} c_{\mathrm{logic}}(\lambda). \tag{18}$$

For the logic constraints used in our DSL, the complexity score is assigned as:

$$c_{\mathrm{logic}}(\lambda) = \begin{cases} 2, & \text{for pairwise constraints, e.g., } \texttt{SAME_SYMBOL}, \texttt{DIFFERENT_SYMBOL}, \\ 2, & \text{for complementarity constraints, e.g., } \texttt{COMPLEMENT_SYMBOL}, \\ 2, & \text{for count constraints, e.g., } \texttt{MIN_SYMBOL_COUNT}, \texttt{MAX_SYMBOL_COUNT}, \\ 3, & \text{for conditional constraints, e.g., } \texttt{IF_POSITION_IS} \Rightarrow \texttt{THEN_POSITION_IS}, \\ 1, & \text{for explicit Boolean connectives, i.e., } \texttt{AND}, \texttt{OR}. \end{cases} \tag{19}$$

The total logic complexity sums these costs over all constraint occurrences and explicit Boolean connectives, including those in nested expressions. Each complete conditional constraint is counted once, including those with a `THEN_POSITION_CANNOT_BE` consequent. Implicit conjunctions between separately listed constraints incur no additional cost.

Table 5: Pattern and logic constraint complexity of baseline methods. Values are means, with 95% confidence intervals in brackets.

Method	Pattern complexity	Logic constraint complexity
Few-shot Claude	22.76 [20.71, 25.03]	3.90 [3.58, 4.23]
Few-shot Gemini	15.63 [13.65, 18.59]	2.88 [2.48, 3.29]
Few-shot GPT	17.99 [16.72, 19.38]	1.43 [0.88, 2.22]
Self-Refinement	16.21 [15.07, 17.39]	2.05 [1.40, 2.84]
Hypothesis Refinement	15.73 [14.60, 16.92]	2.32 [1.52, 3.41]
Claude Code	12.07 [11.33, 12.82]	1.86 [1.49, 2.24]
Codex	11.64 [10.93, 12.37]	0.86 [0.41, 1.41]
GA	16.52 [15.27, 17.88]	1.23 [0.81, 1.70]
SA	6.18 [5.53, 6.86]	1.29 [0.90, 1.73]
LLM-SR	8.57 [6.65, 10.31]	2.17 [1.30, 3.04]

A.2.3 Rule Complexity of Baseline Methods

Table 5 reports the mean complexity of the rules discovered by each baseline. The 95% confidence intervals are estimated using percentile bootstrap with 10,000 resamples over the available evaluable rules for each method. In general, lower rule complexity is preferred. Together with the experimental results shown in Section 4, these findings suggest that coding agents achieve a favorable balance between overall performance and rule complexity.

A.3 Empirical Justification of BoundaryAcc

Section 3.4 of the main text introduces *boundary accuracy* ($\mathrm{BoundaryAcc}$) as a boundary-faithful evaluation metric for symbolic sequence rule discovery, and the formal definition is given in Section A.2.1. This appendix uses the same notation as Appendix A.2.1: $\mathcal{G}$ denotes the ground-truth (GT) rule, $\mathcal{R}$ the learned rule, $\mathcal{P}_{\mathcal{G}}$ the set of generated GT boundary-crossing (BC) pairs, and $\mathcal{B}_{\mathcal{G}}$, $\mathcal{B}_{\mathrm{covered}}$ the boundary-element sets defined in Eq. 13 and Eq. 14. We complement the definition with a controlled empirical study that quantifies four properties of the metric: (**A**) *correctness* – $\mathrm{BoundaryAcc}$ behaves exactly as predicted on the identity case and on logically equivalent rewrites of the GT rule; (**B**) *stability* – the Monte-Carlo estimator of $\mathrm{BoundaryAcc}$ has limited sensitivity to the random seed of the boundary-crossing sampler, the number of positive seed sequences, and the two main hyper-parameters $M = \texttt{max_pairs_per_seq}$ and $V = \texttt{max_variants_per_boundary}$; (**C**) *robustness* – $\mathrm{BoundaryAcc}$ decreases overall with controlled rule corruption and is exactly invariant to test-set label noise; (**D**) *discriminative power* – $\mathrm{BoundaryAcc}$ separates predicted rules that aggregate-level metrics (Acc, F1) cannot tell apart, and is invariant to a change of the negative-to-positive ratio in the test set.

A.3.1 Experimental Roadmap and Setup

Roadmap. The controlled validation answers the following questions. **Q1: Correctness** – does BoundaryAcc equal one when the candidate rule is semantically identical to the GT rule (Section A.3.2 and Table 9)? **Q2: Estimator stability** – how much does BoundaryAcc move when the boundary-crossing sampler changes seed, sample size, or hyper-parameters (Section A.3.3 and Figure 10)? **Q3: Robustness** – does BoundaryAcc decrease when the rule is semantically corrupted, and does it remain stable under stored-label noise (Section A.3.4, Figure 11, and Figure 12)? **Q4: Complementarity with aggregate metrics** – can BoundaryAcc distinguish rules that Accuracy or F1 group together, and is it immune to class-ratio manipulation (Section A.3.5, Figure 13, and Figure 14)? Table 6 gives the detailed protocol and headline result for Q1–Q4, while Table 7 reports the full all-splits validation.

How to read the all-splits table. Table 7 is the primary quantitative evidence in this appendix. The column "A eval" counts rules for which the generator produced at least one valid GT boundary-crossing pair in the identity test. Across the five protein splits, all 50 rules are evaluable; across the four non-empty virtual1 splits, some rules have no valid generated boundary-crossing pairs under the current generator, so their BoundaryAcc is marked as missing and Coverage is zero. This is not counted as evidence against the correctness theorem: the theorem applies when $\mathcal{P}_{\mathcal{G}} \neq \emptyset$. Instead, it demonstrates why Coverage is needed: it tells the reader whether a BoundaryAcc value is supported by adequate boundary coverage. The remaining columns summarise the main effect sizes: seed

Table 6: Roadmap-level summary of the controlled BoundaryAcc validation experiments. Each row corresponds to one experimental question in Sections A.3.2–A.3.5.

Topic	Question / claim tested	Protocol	Main result
Identity & rewrite	Does BoundaryAcc equal one for a semantically identical rule?	Evaluate GT and equivalent rewrites on all 90 rules.	All 73 evaluable identity cases reach 1.000; easy-protein mean coverage is 0.802.
Estimator stability	Is the estimator stable across sampling settings?	Sweep sampler seeds and budgets N, M, and V.	Maximum average per-rule seed std: 0.041 across nine splits.
Rule corruption	Does BoundaryAcc detect semantic corruption?	Apply $k = 0, \ldots, 5$ perturbation steps.	Easy-protein BoundaryAcc: 1.000 to 0.020; Accuracy at $k = 5$: 0.700.
Label noise	Is BoundaryAcc invariant to stored-label noise?	Flip a fraction $\varepsilon \in \{0, .05, .10, .20, .30\}$ within each class.	Easy-protein BoundaryAcc stays 0.446; Accuracy falls 0.841 to 0.638.
Acc-tied buckets	Does BoundaryAcc separate candidates with similar Accuracy?	Bucket 400 candidates per split within each GT rule, using width 0.005.	Maximum spread: 0.928 (easy protein), 0.926 (easy virtual1).
Class imbalance	Is BoundaryAcc invariant to the test class ratio?	Vary $\rho \in \{1, 2, 5, 10, 20, 50, 100\}$ for fixed candidates.	Easy-protein Accuracy: 0.725 to 0.988; BoundaryAcc stays 0.446.

Table 7: Controlled validation on 90 GT rules across nine non-empty splits. "A eval" counts identity cases with at least one valid BC pair. "B std" is the largest, over $k \in \{1, 2, 3\}$, average per-rule standard deviation across ten sampler seeds. "C BC0/BC5" reports mean BoundaryAcc at corruption steps 0 and 5. "Noise ΔAcc" is Accuracy at noise 0 minus Accuracy at noise .3; "Noise ΔBC" is the BoundaryAcc range. "D max spread" is computed within Accuracy buckets of width .005 separately for each GT rule. "D-imb ΔAcc/BC" reports the Accuracy gain and BoundaryAcc range across class ratios. Both nuisance sweeps reuse fixed BC pools across nuisance levels.

Domain	Diff.	Rules	A eval	B std	C BC0	C BC5	Noise ΔAcc	Noise ΔBC	D max spread	D-imb ΔAcc/BC
protein	complex	10	10	0.014	1.000	0.050	0.272	0.000	0.975	0.295/0.000
protein	easy	10	10	0.038	1.000	0.020	0.204	0.000	0.928	0.263/0.000
protein	hard	10	10	0.013	1.000	0.136	0.274	0.000	1.000	0.341/0.000
protein	medium	10	10	0.014	1.000	0.178	0.201	0.000	0.992	0.337/0.000
protein	simple	10	10	0.010	1.000	0.023	-0.046	0.000	0.181	0.423/0.000
virtual1	easy	10	5	0.038	1.000	0.093	0.259	0.000	0.926	0.185/0.000
virtual1	hard	10	4	0.016	1.000	0.055	0.299	0.000	1.000	0.245/0.000
virtual1	medium	10	5	0.029	1.000	0.105	0.254	0.000	0.967	0.284/0.000
virtual1	simple	10	9	0.041	1.000	0.079	0.138	0.000	0.214	0.222/0.000

variability (B std), corruption response (C BC0/BC5), label-noise robustness (Noise ΔAcc vs. Noise ΔBC), Acc-tied separation (D max spread), and class-imbalance sensitivity (D-imb ΔAcc/BC). Figure 8 visualises the same information graphically: panel (a) corresponds to the "A eval" column, (b) to the C corruption columns, (c) to the noise/imbalance columns, and (d) to "D max spread".

Datasets. We use the synthetic data generator of SeqRule-Bench (Section 3.3 of the main paper) to instantiate controlled splits in which the GT rule is known by construction. Two synthetic domains are used: the *protein* domain uses the standard 20-amino-acid alphabet, and the *virtual1* domain uses a 26-letter alphabet. For each domain we use five difficulty levels labelled *simple*, *easy*, *medium*, *hard*, and *complex*; difficulty is controlled by the number of conserved pattern positions, the number of variable-length gaps, and the number of logic clauses in the GT rule, so harder splits have more conserved positions, longer wildcards, and more relational constraints. Each (domain, difficulty) pair defines one *split*, and we refer to splits by their two attributes – e.g. "the easy protein split" or "the complex virtual1 split". Each split contains 10 GT rules with stored training and test samples. Metadata-only entries without sample files are not treated as evaluated rules. Per rule, the benchmark synthesizer emits a training set and a held-out test set drawn from the same distribution, where positives are sequences that satisfy the rule and negatives are sampled by the four-layer hard-negative miner of Section 3.3 (random sequences, pattern-rejected sequences, logic-rejected sequences, and minimal-change sequences). Per-rule sample counts are not balanced and vary substantially with difficulty: typical per-rule training sets contain on the order of 10 to 1,600 positive sequences and 1,000 to 1,500 negative sequences, while held-out test sets contain on the order of 1 to 400 positive sequences and 200 to 400 negative sequences. Most splits are therefore strongly negative-dominated

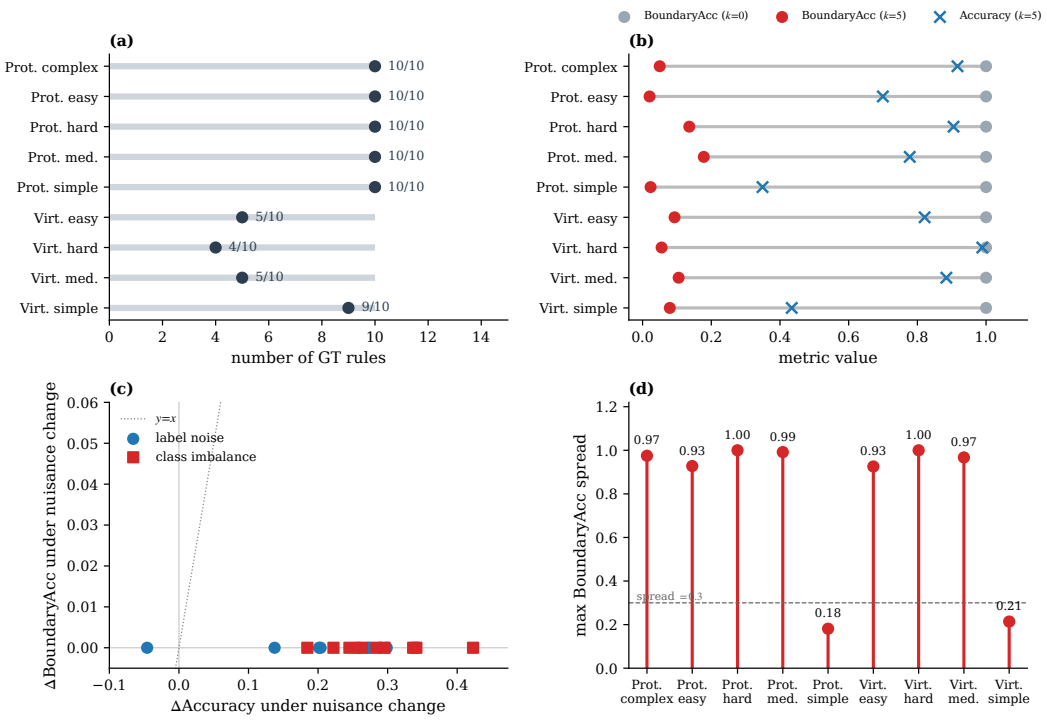


Figure 8: All-splits validation on 90 GT rules. (a) Number of evaluable identity cases; cases with no generated boundary-crossing (BC) pairs are undefined. (b) Under corruption, mean BoundaryAcc falls from 1.000 to 0.020–0.178, while final Accuracy stays in 0.349–0.989. (c) Changing stored labels or test class ratios affects Accuracy but leaves BoundaryAcc unchanged when the BC pools and candidate rules are fixed. (d) The largest within-GT Accuracy-tied spread exceeds 0.3 in 7 of nine splits. These results support BoundaryAcc as a complementary measure of boundary agreement.

(test positive-to-negative ratios in our scope range from 0.00 to 2.03, with 7 of 9 non-empty splits below 0.6); this imbalance is the intentional shape of the benchmark and is what makes aggregate metrics exhibit the floor effects analysed in Section A.3.4. The complex virtual1 split contains 0 rules in the benchmark instance we use and is therefore excluded. This gives 9 non-empty splits and 90 GT rules, which is the full empirical scope of this appendix. The same rule executor (defined in the next paragraph) is used to evaluate both the GT rule and the candidate rule, so any metric difference we report is caused by rule semantics rather than by evaluator mismatch.

Notation. We follow the symbols defined in Appendix A.2.1: $\mathcal{G}$ is the ground-truth rule, $\mathcal{R} = (e, \Lambda)$ is the learned (or candidate) rule with symbolic pattern e and logic-clause set Λ, $\mathcal{R}(x) \in \{0, 1\}$ is the indicator returned on sequence x by the *rule executor* (the deterministic matcher that decides whether a sequence satisfies a rule and returns 1 if it does), and $\mathcal{P}_{\mathcal{G}}$ is the set of generated boundary-crossing pairs. D denotes the evaluation dataset used for a given rule, D^+ and D^- denote its positive and negative subsets, and we write $D^+_{r,\text{train}}$ (resp. $D^+_{r,\text{test}}$) for the training (resp. test) positives of rule r. For convenience we make the directional scoring used in this appendix explicit, retain the coverage definition of Eq. 14, and report one derived quantity:

$$\begin{aligned} \text{BoundaryAcc}(\mathcal{R}, \mathcal{G}, D) &= \frac{1}{|\mathcal{P}_{\mathcal{G}}|} \sum_{(x,x') \in \mathcal{P}_{\mathcal{G}}} c_{\mathcal{R},\mathcal{G}}(x, x'), \\ c_{\mathcal{R},\mathcal{G}}(x, x') &= \mathbf{1}\begin{bmatrix} \mathcal{G}(x) = 1 \ \wedge \ \mathcal{G}(x') = 0 \\ \wedge \ \mathcal{R}(x) = 1 \ \wedge \ \mathcal{R}(x') = 0 \end{bmatrix}, \\ \text{BoundaryAcc}_{\text{eff}} &= \text{BoundaryAcc} \cdot \text{Coverage}. \end{aligned} \tag{20}$$

After generating these samples, we randomly shuffle them and swap the order of the two sequences within each pair (x, x'), yielding both positive-to-negative and negative-to-positive pairs.

The denominator remains the complete set of valid GT boundary pairs. We do not pre-filter pairs by $\mathcal{R}(x) = 1$: a candidate that misses a GT-positive seed is therefore penalized rather than silently removed from the evaluation. This criterion distinguishes the desired $1 \to 0$ transition from the reversed $0 \to 1$ transition, which is not counted as consistent. $\mathcal{B}_\mathcal{G}$ enumerates the boundary-relevant components of $\mathcal{G}$ (conserved pattern positions or character classes, wildcard/gap-length constraints, and logic clauses such as SAME_SYMBOL and DIFFERENT_SYMBOL). Each pair $(x, x') \in \mathcal{P}_\mathcal{G}$ is produced by changing one or more such elements; let $\mathcal{B}(x, x') \subseteq \mathcal{B}_\mathcal{G}$ be the subset of elements perturbed by that pair. Then $\mathcal{B}_{\text{covered}} = \bigcup_{(x,x') \in \mathcal{P}_\mathcal{G}} \mathcal{B}(x, x')$ and

$$\text{Coverage}(\mathcal{P}_\mathcal{G}, \mathcal{G}) \;=\; \frac{|\mathcal{B}_{\text{covered}}|}{|\mathcal{B}_\mathcal{G}|}. \tag{21}$$

Coverage is therefore a diagnostic for whether the BoundaryAcc estimate has tested the whole rule or only a subset of its boundary conditions. A high BoundaryAcc with low Coverage means that the candidate behaves well on the generated pairs, but the generated pairs do not yet cover all rule elements. When no valid boundary-crossing pair can be generated ($\mathcal{P}_\mathcal{G} = \emptyset$), we report BoundaryAcc as missing and Coverage as 0.

Aggregate metrics. For every (GT rule, candidate rule) pair we evaluate the candidate on the rule's held-out test set $D_{r,\text{test}}$ and record the standard binary-classification metrics

$$\begin{aligned} \text{Acc} &= (TP + TN)/(TP + TN + FP + FN), & \text{F1} &= 2\,TP/(2\,TP + FP + FN), \\ \text{PPV} &= TP/(TP + FP), & \text{TPR} &= TP/(TP + FN), \\ \text{FPR} &= FP/(FP + TN), & \text{FNR} &= FN/(TP + FN), \end{aligned}$$

where TP, TN, FP, FN are the counts of true positives, true negatives, false positives and false negatives in the test set (positives and negatives defined by the stored test labels), and PPV, TPR, FPR, FNR are respectively the positive predictive value, true positive rate, false positive rate and false negative rate. boundary metrics (BoundaryAcc, Coverage, $\text{BoundaryAcc}_{\text{eff}}$) are computed against the GT rule $\mathcal{G}$ and candidate rule $\mathcal{R}$ via the generator described next, and never read the stored test labels. Each figure and table in this appendix is produced directly from the recorded metric outputs of the corresponding experiment, with no post-hoc selection. We report both domains when the trend differs, and use the easy protein split as the default per-rule view when both domains support the same conclusion.

Boundary-crossing pair generator. $\mathcal{P}_\mathcal{G}$ is built by a deterministic procedure with three sampling hyper-parameters: the number of positive seed sequences N, the maximum number of pairs returned per seed $M =$ `max_pairs_per_seq` (counted across all boundary elements, not per element), and the maximum number of candidate variants tried for each boundary element $V =$ `max_variants_per_boundary`. $V = 3$ by default and is varied in the sensitivity experiment; N is capped at 80 but is reduced to the number of available training positives whenever a rule has fewer than 80 of them, so the effective N for rule r is $N_r = \min(80, |D^+_{r,\text{train}}|)$; M varies by experiment (see the table at the end of this paragraph). The procedure is:

1. Enumerate the boundary-element set $\mathcal{B}_\mathcal{G}$ of $\mathcal{G}$. Each conserved pattern position contributes one element, each variable-length gap contributes one element (its lower and upper endpoints), and each logic clause contributes one element.
2. Sample N_r positive seed sequences uniformly without replacement from the rule's training positives.
3. For every seed x_i and every $b \in \mathcal{B}_\mathcal{G}$, draw at most V single-element edits of x_i that change the way b is matched. Concretely: for a conserved position, replace the residue with up to V alternative residues from the alphabet; for a gap endpoint, increment or decrement the gap length by ± 1; for a logic clause, edit one of the residues coupled by that clause so the clause flips truth value. Because x_i is a GT-positive seed, keep the candidate x' if and only if $\mathcal{G}(x_i) = 1$ and $\mathcal{G}(x') = 0$, and record the pair (x_i, x') together with $\mathcal{B}(x_i, x') = \{b\}$.
4. For each seed x_i, retain at most M pairs by drawing a uniform random subset of the valid candidates across all of $\mathcal{B}_\mathcal{G}$ when the cap is exceeded. The truncation is per seed, so the per-seed pair count is $|\mathcal{P}_\mathcal{G}(x_i)| \le M$.

Table 8: Sampler budget (N, M, V) per experiment. N is the cap on positive seed sequences (the effective value is $N_r = \min(N, |D^+_{r,\text{train}}|)$); M is the per-seed pair limit; V is the per-boundary-element variant limit.

Experiment	N cap	M	V
Correctness (Section A.3.2)	80	10	3
Stability (Section A.3.3)	80 (or swept)	5 (or swept)	3 (or swept)
Corruption / label noise (Section A.3.4)	80	8 / 10	3
Discriminative power (Section A.3.5)	80	8	3

5. Return $\mathcal{P}_\mathcal{G} = \bigcup_{i=1}^{N_r} \mathcal{P}_\mathcal{G}(x_i)$.

The size of $\mathcal{P}_\mathcal{G}$ is therefore bounded by $|\mathcal{P}_\mathcal{G}| \leq N_r \cdot M$. For example, with $M = 10$ and a rule whose training positives saturate the cap so $N_r = 80$, the upper bound is 800 (matching the high end of $|\mathcal{P}_\mathcal{G}|$ in Table 9); for a rule with $|D^+_{r,\text{train}}| = 19$ we have $N_r = 19$ and $|\mathcal{P}_\mathcal{G}| \leq 190$. Total work is bounded by $O(N_r \cdot |\mathcal{B}_\mathcal{G}| \cdot V)$ rule executions per GT rule, which keeps the generator cheap (under one CPU-second per rule in the original setup). Randomness enters through positive-seed selection, boundary-edit sampling, and pair subsampling; all are fixed by the sampler seed in this rerun.

The per-experiment value of M used below is summarised in Table 8.

Candidate (predicted) rule generation. Predicted rules in this appendix are not learned by a model. They are produced by a fixed catalogue of **six controlled rule-perturbation operators** (described next) acting on $\mathcal{G}$. Replacing a rule learner with a deterministic perturbation operator removes algorithm-specific confounders: any failure mode reported below is therefore a property of the metric itself rather than of a particular rule learner. Applying $k \in \{0, 1, 2, 3, 4, 5\}$ randomly chosen operators in sequence to $\mathcal{G}$ yields a candidate rule $\mathcal{R}_k$. The special case $k = 0$ leaves the rule unchanged and is used as the identity case in Section A.3.2. Different candidates with the same k correspond to different random selections of operators and operator targets; we use 8 random seeds per k in the discriminative-power experiments.

Rule-perturbation operators. The catalogue contains the six operators listed below. Each preserves the syntactic well-formedness of the resulting rule, and the resulting candidate is parsed by the same rule executor that defines $\mathcal{G}$. Examples assume a toy GT rule `A-C-D-x{1,3}-K` with logic clause SAME_SYMBOL$(2, 5)$.

- CHANGE-CONSERVED – replace the residue at one conserved pattern position with a different residue from the alphabet (`A-C-D-x{1,3}-K` → `A-C-E-x{1,3}-K`).
- DROP-CONSERVED – replace the residue at one conserved pattern position with a wildcard (`A-C-D-x{1,3}-K` → `A-C-x-x{1,3}-K`).
- SHIFT-WILDCARD – shift one endpoint of a variable-length gap by ± 1 (`x{1,3}` → `x{2,3}`).
- DROP-LOGIC – remove one logic clause from Λ (remove SAME_SYMBOL$(2, 5)$).
- FLIP-SAME/DIFF – replace SAME_SYMBOL(i, j) with DIFFERENT_SYMBOL(i, j) (or vice versa).
- ADD-SPURIOUS-SAME – append a new SAME_SYMBOL(i, j) clause linking two pattern positions that were unconstrained in $\mathcal{G}$.

Within one perturbation step, the operator is sampled uniformly at random from the catalogue, and its target (which conserved position, which gap, or which clause) is sampled uniformly at random from the eligible candidates in $\mathcal{G}$. If a sampled operator has no eligible target, that step leaves the rule unchanged. Thus k counts attempted perturbation steps, not necessarily distinct semantic changes.

A.3.2 CORRECTNESS

This experiment tests the formal correctness claim: if the candidate rule has the same decision function as the GT rule, BoundaryAcc should be exactly one. The purpose is to rule out implementation artefacts before testing harder cases: if BoundaryAcc fails on identity or semantic-preserving rewrites, then any later robustness claim would be meaningless.

Table 9: Identity sanity test on the easy protein split: when the predicted rule equals GT, every rule yields $\mathrm{BoundaryAcc} = 1$. *Conclusion.* The correctness property holds for all 10 evaluable easy-protein rules; Coverage $\in [0.73, 1.00]$ confirms that the boundary tested is non-trivial.

Rule	Acc	F1	BoundaryAcc	Coverage	$\mathrm{BoundaryAcc}_{\mathrm{eff}}$	$\lvert\mathcal{P}_{\mathcal{G}}\rvert$
PR-001	1.000	1.000	1.000	0.727	0.727	190
PR-002	0.991	0.984	1.000	0.727	0.727	800
PR-003	1.000	1.000	1.000	0.778	0.778	800
PR-004	1.000	1.000	1.000	1.000	1.000	800
PR-005	1.000	1.000	1.000	0.889	0.889	800
PR-006	0.974	0.862	1.000	0.889	0.889	632
PR-007	0.998	0.997	1.000	0.727	0.727	800
PR-008	0.992	0.990	1.000	0.778	0.778	800
PR-009	1.000	1.000	1.000	0.778	0.778	800
PR-010	0.979	0.980	1.000	0.727	0.727	784

Protocol. For every GT rule $\mathcal{G}$ we create two candidate rules. First, the identity candidate sets $\mathcal{R} = \mathcal{G}$. Second, the equivalent-rewrite candidate keeps the same pattern and same logic clauses but permutes the order of conjunctive `logic_constraints`. This rewrite changes the surface string but not the decision function. We then run the same generator described in Section A.3.1 (default $N = 80$, $M = 10$, $V = 3$), compute BoundaryAcc and Coverage, and check whether every evaluable case has BoundaryAcc exactly equal to one. The only exclusion criterion is $\mathcal{P}_{\mathcal{G}} = \emptyset$, in which case BoundaryAcc is undefined and Coverage is reported as zero.

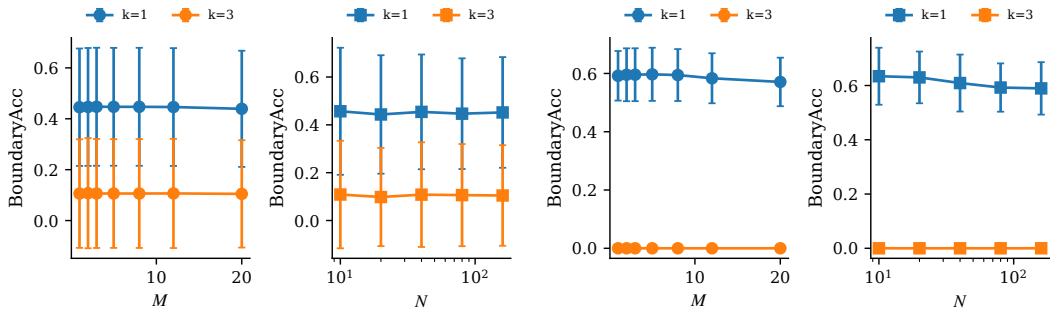
Identity Sanity Setting $\mathcal{R} = \mathcal{G}$ should yield $\mathrm{BoundaryAcc} = 1$. Table 9 reports the metric values on the the easy protein split: all ten rules satisfy $\mathrm{BoundaryAcc} = 1.000$ ($\min = \max = 1.0$), with mean Coverage 0.802. The same conclusion holds on the easy virtual1 split: every evaluable rule has $\mathrm{BoundaryAcc} = 1.000$. Across the full validation set, all evaluable identity cases satisfy $\mathrm{BoundaryAcc} = 1.000$ (Table 7). The fact that several virtual1 rules are not evaluable in the identity test is explicitly reflected by the "A eval" column and by Coverage $= 0$; rules with $\mathcal{P}_{\mathcal{G}} = \emptyset$ indicate that the generator produced no boundary-crossing pairs for that rule rather than a violation of the identity property.

Logically Equivalent Rewrites. Reordering the conjuncts of `logic_constraints` (e.g. swapping two SAME_SYMBOL clauses) does not change the underlying set, hence $\mathcal{R}(x) = \mathcal{G}(x)$ for all x. All evaluable rewrites continue to score $\mathrm{BoundaryAcc} = 1.000$ ($10/10$ on the easy protein split; $3/3$ evaluable cases on the easy virtual1 split; see the "Identity & rewrite" row of Table 6). The metric is therefore invariant to the surface syntax of the rule and depends only on its semantic decision function.

Monte-Carlo Convergence. *Purpose.* BoundaryAcc is estimated from sampled boundary-crossing pairs, so we need to verify that the estimate is not an artefact of an arbitrary sample budget. We therefore vary the two practical sampling knobs while holding the underlying pair $(\mathcal{G}, \mathcal{R})$ fixed.

Protocol. We construct two corrupted candidates for each GT rule: a mild candidate produced by one perturbation step ($k = 1$) and a stronger candidate produced by three perturbation steps ($k = 3$). We then sweep M, the maximum number of BC pairs generated per positive sequence, and N, the number of positive seed sequences used to start generation. The reported curves average across rules and random seeds; the expected result is that the curve should stabilise quickly rather than drifting with larger budgets.

Figure 9 shows the estimator under changes to M and N. On easy protein, the $k = 1$ mean changes from 0.445 to 0.439 as M grows from 1 to 20; the $k = 3$ endpoints are 0.106 and 0.105. Across $N = 10$–160, the easy-protein $k = 1$ means range from 0.443 to 0.457. On easy virtual1, the $k = 1$ mean changes from 0.592 to 0.571 over the M sweep. These estimates are relatively stable over the tested budgets, although stability of split means does not establish convergence for every individual rule.



(a) Easy protein split

(b) Easy virtual1 split

Figure 9: Monte-Carlo sensitivity on the two easy splits. Left: per-sequence pair budget M; right: positive-seed budget N. Each mean and standard-deviation bar pools evaluable rules and five sampler seeds. The bars therefore include between-rule heterogeneity, not just Monte-Carlo uncertainty. The split means change modestly over the tested budgets.

A.3.3 Stability (Internal Robustness)

This experiment holds the candidate rule fixed and varies only the sampling procedure. It therefore isolates estimator noise from rule-quality differences.

Protocol. For each GT rule we first generate a fixed corrupted candidate $\mathcal{R}_k$ using $k \in \{1, 2, 3\}$ perturbation steps. We then repeat BoundaryAcc estimation under three controlled changes: (i) resampling the set with different random seeds, (ii) changing the number of positive seed sequences N, and (iii) changing the generator hyper-parameters M and V, where M limits the number of pairs per positive sequence and V limits the number of candidate variants tried around each boundary element. Because $\mathcal{R}_k$ itself is fixed within each comparison, any movement in BoundaryAcc is estimator variance rather than a change in rule quality.

Expected behaviour. A usable metric should have small seed variance and should not require exhaustive boundary-crossing enumeration. We therefore expect the seed standard deviation to be small and the N, M, V curves to plateau at moderate values.

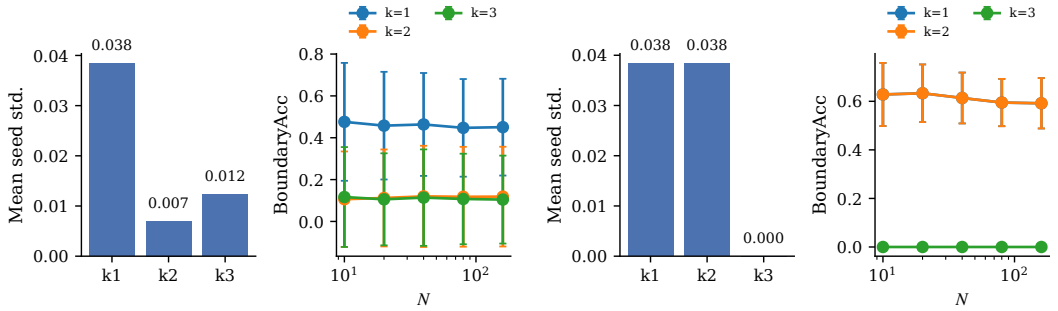
Seed sensitivity. With the candidate fixed, the average per-rule standard deviations across ten sampler seeds are 0.038, 0.007, 0.012 for $k = 1, 2, 3$ on easy protein and 0.038, 0.038, 0.000 on easy virtual1. The largest such value over all nine splits is 0.041 (Table 7).

Sample-size sensitivity (N). Figure 10 (right) varies the positive-seed budget. For $N = 10, 40, 80, 160$, the easy-protein $k = 1$ means are 0.476, 0.464, 0.447, 0.451; the corresponding easy-virtual1 means are 0.628, 0.614, 0.595, 0.592. These split averages show modest budget sensitivity; they do not guarantee the same stability for every GT rule.

Hyper-parameter sensitivity (M and V). For $k = 1$, increasing M from 1 to 20 changes mean BoundaryAcc from 0.449 to 0.438 on easy protein and from 0.595 to 0.573 on easy virtual1. Increasing V from 1 to 5 changes the means from 0.462 to 0.471 and from 0.559 to 0.600, respectively. The V sweep uses the first N available positive seeds and repeats boundary-edit sampling with three seeds. These checks quantify sensitivity to the sampled boundary distribution rather than proving exact invariance to generator settings.

A.3.4 Robustness to Rule Corruption and Label Noise

Response to controlled rule corruption. *Purpose.* A boundary-faithful rule metric should react when the candidate rule is progressively moved away from GT. Standard accuracy can fail here because most test sequences are negatives far from the decision boundary; a broken rule can still classify many far negatives correctly.



(a) Easy protein split

(b) Easy virtual1 split

Figure 10: Estimator sensitivity. Left: average per-rule standard deviation across ten sampler seeds, at most 0.038 on easy protein and 0.038 on easy virtual1. Right: means and standard deviations over evaluable rules and three sampler seeds at each N. Boundary sampling contributes limited variation in these aggregate comparisons, while the right-hand error bars also reflect differences between rules.

Table 10: Controlled corruption on easy protein. Directional BoundaryAcc falls overall from 1.000 to 0.020, whereas final accuracy remains 0.700. Each row averages five perturbation seeds per rule and then averages the ten rules; the BC sampler seed is fixed.

k	Acc	F1	BoundaryAcc	$\text{BoundaryAcc}_{\text{eff}}$
0	0.993	0.981	1.000	0.802
1	0.765	0.273	0.284	0.229
2	0.717	0.203	0.242	0.194
3	0.688	0.173	0.181	0.146
4	0.697	0.030	0.012	0.009
5	0.700	0.033	0.020	0.015

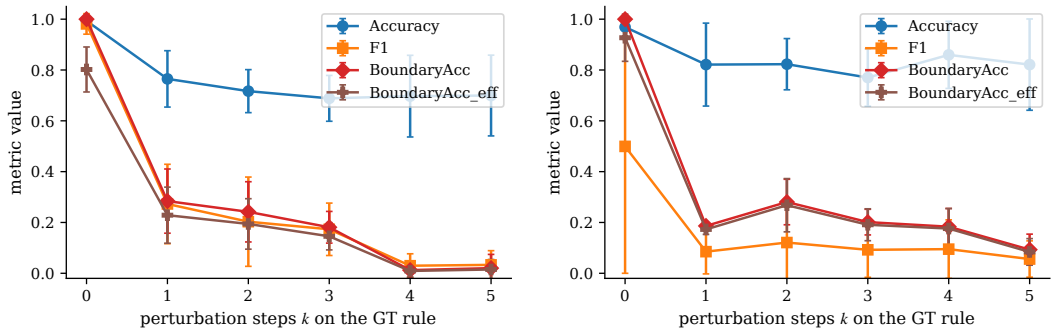
Protocol. Starting from each GT rule, we generate candidates $\mathcal{R}_k$ by applying $k = 0, 1, 2, 3, 4, 5$ perturbation steps using the six operators defined in Section A.3.1. At $k = 0$ the candidate is the GT rule; as k increases, the candidate loses conserved symbols, shifts gap constraints, drops logic, or adds/removes relational constraints. We then compute Accuracy, F1, BoundaryAcc, and $\text{BoundaryAcc}_{\text{eff}}$ for every k. The expected pattern is $\text{BoundaryAcc}(k = 0) = 1$ followed by a large decrease as k grows; if Accuracy plateaus, that is evidence that aggregate test-set metrics are missing boundary degradation.

Figure 11 and Table 10 show a strong overall decline in directional agreement under corruption. For $k = 0, \ldots, 5$, easy-protein BoundaryAcc is 1.000, 0.284, 0.242, 0.181, 0.012, 0.020. Final Accuracy remains 0.700. Easy-virtual1 BoundaryAcc decreases from 1.000 to 0.093, with final Accuracy 0.821. All nine splits show a BoundaryAcc drop of at least 0.822. The curves need not be strictly monotone: the sampled perturbations can have different semantic effects, including redundant edits. A small rebound alone is not evidence that sampling noise is its cause. The contrast with the negative-dominated Accuracy floor supports BoundaryAcc as a complementary corruption signal.

The previous experiment showed that rule-level corruption is reflected clearly in BoundaryAcc. We next separate test-set nuisance from rule-quality degradation: label noise should hurt label-driven metrics but leave a boundary metric untouched.

Invariance to label noise. *Purpose.* BoundaryAcc is defined against the GT decision function, not against stored test labels. Label noise in the test file should hurt accuracy and F1 but should not change BoundaryAcc, as long as the GT rule and candidate rule are unchanged.

Protocol. For each GT rule we fix a moderately corrupted candidate ($k = 1$) and flip a fraction $\varepsilon \in \{0, 0.05, 0.10, 0.20, 0.30\}$ of test labels by uniformly selecting a rounded fraction ε of each class and flipping their labels, then recompute accuracy and F1 on the noisy labels. BoundaryAcc is recomputed using the original GT rule and the same 1-step perturbed candidate; the label file is not consulted by the boundary-crossing pair generator. The expected pattern is a monotone degradation in accuracy/F1 and an essentially flat BoundaryAcc curve.



(a) Easy protein split

(b) Easy virtual1 split

Figure 11: Response to controlled rule corruption. BoundaryAcc decreases strongly from its identity value, while accuracy remains comparatively high on the negative-dominated test sets. F1 also detects substantial degradation, so this experiment supports a complementary boundary measure rather than a claim that F1 is insensitive. Error bars are standard deviations across per-rule means. Neither strict monotonicity nor a linear response is assumed.

Table 11: Robustness to test-set label noise on the easy protein split. BoundaryAcc is invariant in ε because it is computed against the GT decision function rather than the (now corrupted) labels.

ε	Acc	F1	BoundaryAcc
0.00	0.841	0.571	0.446
0.05	0.806	0.512	0.446
0.10	0.771	0.463	0.446
0.20	0.703	0.385	0.446
0.30	0.638	0.324	0.446

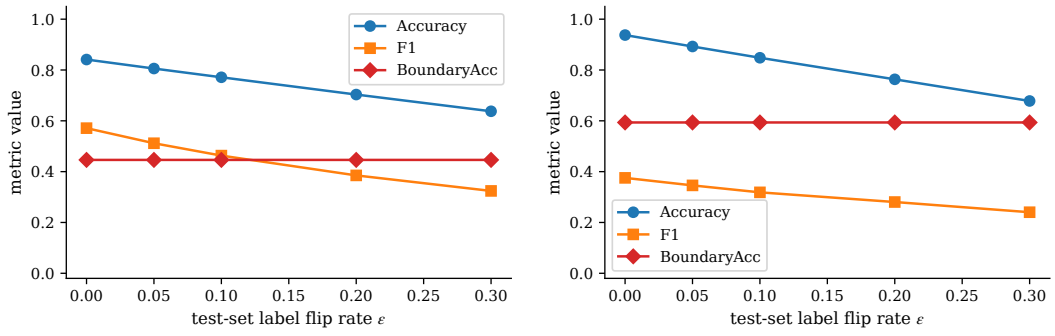
Result. With the GT rule, candidate, positive seeds, and BC pool fixed, stored-label noise cannot change BoundaryAcc. The rerun uses identical sampler seeds across noise levels, so the invariance is exact. Easy-protein BoundaryAcc stays at 0.446 while accuracy falls from 0.841 to 0.638; easy-virtual1 BoundaryAcc stays at 0.594. All nine splits have zero BoundaryAcc range across noise levels (Table 7). Small drifts in independently resampled evaluations reflect changes in sampled pairs, not dependence on stored labels.

A.3.5 Discriminative Power vs. Acc / F1 / Precision / Recall

Accuracy-tied groups with non-trivial BoundaryAcc spread. *Purpose.* Accuracy is a scalar average over a fixed test set. If two candidate rules have nearly identical accuracy but very different boundary behaviour, then accuracy is missing an important dimension of rule quality. This experiment searches for exactly those cases.

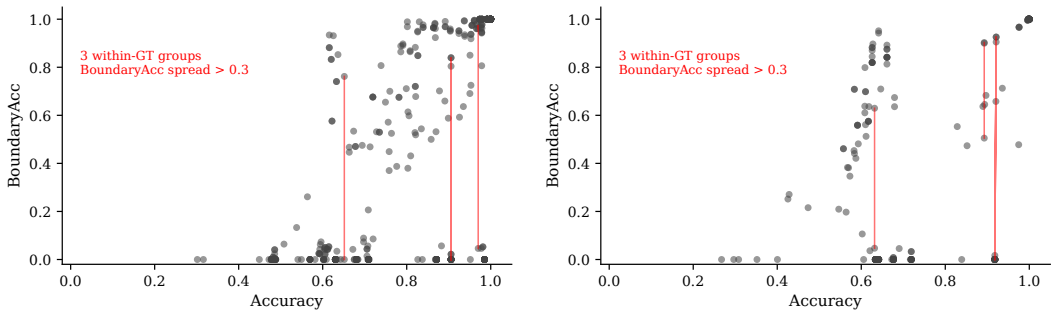
Protocol. For every GT rule we generate a population of candidate rules by applying $k \in \{0, 1, 2, 3, 4\}$ perturbation steps with 8 random seeds per k, giving 40 candidates per GT rule and therefore 400 candidates per domain (since each split has 10 rules). Each candidate is scored by accuracy and BoundaryAcc. Within each GT rule separately, we then bucket candidates whose accuracy values have the same rounded $\mathrm{Accuracy}/0.005$ value and measure the within-bucket BoundaryAcc spread (i.e. $\max \mathrm{BoundaryAcc} - \min \mathrm{BoundaryAcc}$). A large within-bucket spread means that two rules are effectively tied under accuracy but have very different boundary-crossing behaviour.

Result. On easy protein, 7 within-GT accuracy buckets have BoundaryAcc spread above 0.1 and 3 exceed 0.3; the largest spread is 0.928. On easy virtual1, these counts are 4 and 3, with maximum spread 0.926. Across the nine splits, the maximum spread is 1.000. Thus similar aggregate accuracy can coexist with very different directional boundary agreement. This comparison concerns similar accuracy values, not identical per-sequence predictions.



(a) Easy protein split (b) Easy virtual1 split

Figure 12: Stored-label noise on the easy splits. Accuracy and F1 decrease as the flipped fraction increases, while directional BoundaryAcc remains 0.446 (protein) and 0.594 (virtual1). The same BC pools are reused across noise levels, isolating stored-label changes from Monte-Carlo variation.



(a) Easy protein split (b) Easy virtual1 split

Figure 13: Accuracy and directional BoundaryAcc for 400 candidates per easy split. Each point is a candidate with defined BoundaryAcc. Red segments connect candidates from the same GT rule and the same rounded accuracy bucket (width .005) with BoundaryAcc spread above .3: 3 groups on easy protein and 3 on easy virtual1. Maximum spreads are 0.928 and 0.926, respectively.

Rank correlation with a proxy oracle. *Purpose.* The Acc-tied analysis above is a local tied-case check. We add a global sanity check: do the metrics broadly agree with a known perturbation-based ordering of the synthetic candidates?

Protocol. For each candidate we know how many perturbation steps were applied to obtain it. We define the `(pattern, logic)` signature of a rule as the pair (p, ℓ), where p is the rule's pattern string written in the domain-specific language (DSL) syntax of SeqRule-Bench (e.g. "`[AILV]-x(2)-[STNQCY]-x(3)-[AILV]`") and ℓ is the multi-line logic-clause block of the rule (one `SAME_SYMBOL` / `DIFFERENT_SYMBOL` / etc. clause per line, normalised by stripping leading/trailing whitespace). We then define $\mathrm{LD}(\mathcal{R}, \mathcal{G}) = L_{\mathrm{char}}(p_{\mathcal{R}}, p_{\mathcal{G}}) + L_{\mathrm{char}}(\ell_{\mathcal{R}}, \ell_{\mathcal{G}})$, where $L_{\mathrm{char}}(\cdot, \cdot)$ is the character-level Levenshtein distance, so LD is the sum of the pattern-string distance and the logic-block distance. The proxy oracle is $q(\mathcal{R}) = -10k - \mathrm{LD}(\mathcal{R}, \mathcal{G})$, with larger meaning closer to GT. We compute the Spearman correlation between q and each metric separately per GT rule and report the mean. This syntactic proxy need not coincide with semantic rule quality; the experiment is a consistency check, not the primary argument for boundary evaluation.

Result. Mean Spearman correlations on easy protein are 0.588 (accuracy), 0.549 (F1), and 0.533 (BoundaryAcc); on easy virtual1 they are 0.411, 0.274, and 0.261. Averaging the nine split means gives 0.512, 0.483, and 0.487, respectively. The positive average correlations support broad agreement with this syntactic perturbation proxy, without establishing BoundaryAcc as the best ranking metric. To retain the original ranking protocol, undefined BoundaryAcc values are assigned zero for this diagnostic, and constant score vectors receive correlation zero; undefined BoundaryAcc is excluded from the other metric means.

Table 12: Mean and standard deviation of per-GT Spearman correlations with the syntactic proxy $-10k - \mathrm{LD}$. This is a complementary ranking diagnostic, not evidence that BoundaryAcc dominates accuracy or F1. The retained handling of undefined and constant scores is stated in the text.

metric	protein mean	std	virtual1 mean	std
Accuracy	$+0.588$	0.075	$+0.411$	0.261
F1	$+0.549$	0.054	$+0.274$	0.276
BoundaryAcc	$+0.533$	0.067	$+0.261$	0.262
$\mathrm{BoundaryAcc}_{\mathrm{eff}}$	$+0.533$	0.067	$+0.261$	0.262

Table 13: Class-imbalance stress test on the easy protein split. Increasing the negative-to-positive ratio ρ inflates accuracy by tens of percentage points, while BoundaryAcc remains essentially constant.

ρ	Acc	F1	BoundaryAcc
1	0.725	0.576	0.446
2	0.814	0.573	0.446
5	0.903	0.564	0.446
10	0.944	0.551	0.446
20	0.967	0.530	0.446
50	0.983	0.486	0.446
100	0.988	0.441	0.446

Class-imbalance stress test. *Purpose.* Accuracy can be inflated by adding many easy negatives to the test set. A decision-boundary metric should not change when the same candidate rule is evaluated under a different negative-to-positive ratio, because the GT boundary itself has not moved.

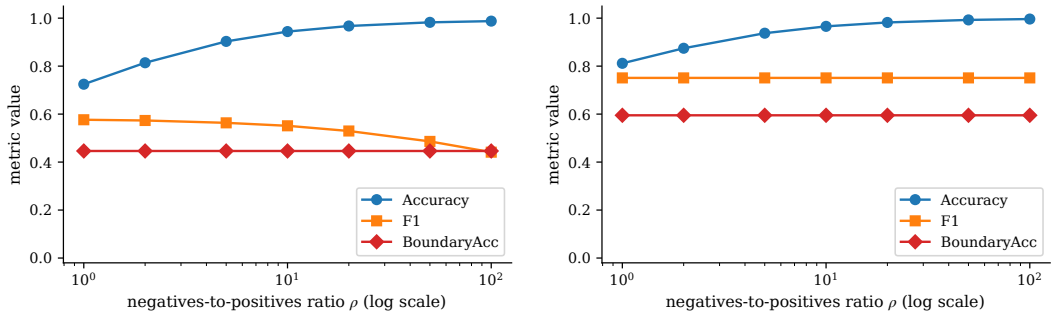
Protocol. We fix one $k = 1$ candidate per GT rule and keep the positive test sequences as they are. Let ρ_0 be the negative-to-positive ratio of the rule's original test set. For target ratios $\rho \in \{1, 2, 5, 10, 20, 50, 100\}$, the resampled negative pool is constructed as follows: if $\rho \leq \rho_0$ we draw $\rho \cdot |D^+_{\text{test}}|$ negatives *without replacement* from the original test negatives (subsample); if $\rho > \rho_0$ we retain the original negative pool and append draws with replacement until reaching the target count. Aggregate metrics (accuracy, F1) are recomputed on each resampled test set. BoundaryAcc is recomputed from boundary-crossing pairs generated by the same procedure as in Section A.3.1–it depends only on $(\mathcal{G}, \mathcal{R})$ and on the training positives that seed the generator, not on the resampled test negatives, so it should remain (essentially) constant across ρ.

Result. With one fixed candidate per GT rule, mean accuracy rises from 0.725 to 0.988 on easy protein and from 0.812 to 0.996 on easy virtual1 as the negative-to-positive ratio grows. Directional BoundaryAcc stays at 0.446 and 0.595, respectively. The BoundaryAcc range is zero in every split because the BC pools are held fixed across ratios. Hence a large accuracy gain caused solely by test composition need not indicate improved boundary agreement.

A.3.6 TAKE-AWAYS

- **Correctness.** All 73 evaluable identity cases, and every evaluable equivalent rewrite, attain BoundaryAcc $= 1$. Cases without BC pairs remain undefined.
- **Sampling sensitivity.** The largest average per-rule standard deviation across sampler seeds is 0.041. Budget sweeps support modest sensitivity of split means over the tested settings, not universal convergence of every rule.
- **Corruption and nuisance.** Every split shows a BoundaryAcc drop of at least 0.822 from $k = 0$ to $k = 5$. The response is not necessarily monotone or linear. Stored-label and test-class-ratio changes leave BoundaryAcc exactly invariant when the generated pools are fixed.
- **Complementarity.** Accuracy-tied candidates of the same GT rule can differ in BoundaryAcc by as much as 1.000. BoundaryAcc has positive mean association with the perturbation proxy but does not dominate all aggregate metrics. It provides an additional measure of boundary agreement.

Effect of directional scoring. For an unchanged pool, write n_{ab} for the number of pairs with $\mathcal{R}(x) = a$ and $\mathcal{R}(x') = b$. The former any-flip score is $(n_{10} + n_{01})/N$, while the directional score



(a) Easy protein split

(b) Easy virtual1 split

Figure 14: Class-ratio stress test with one fixed $k = 1$ candidate per GT rule, averaged across rules and three resampling seeds. Accuracy increases from 0.725 to 0.988 on easy protein and 0.812 to 0.996 on easy virtual1. BoundaryAcc stays at 0.446 and 0.595. Reusing the BC pools isolates the effect of test composition.

is n_{10}/N, with the same $N = \sum_{a,b} n_{ab}$. Across the 2920 evaluable candidates in experiment D, the mean score changes from 0.462 to 0.457; 693 candidates lose reverse-direction credit. This paired comparison changes only the numerator. The correctness, nuisance-invariance, corruption-response, and accuracy-tied discrimination conclusions remain supported under the directional definition.

Reproducibility and compute. The rerun preserves the original GT rules, data splits, candidate-perturbation seeds, and metric aggregation. Because the historical generator did not seed its internal PRNG and did not save its BC pools, historical pairs cannot be reconstructed exactly. We regenerate pools with explicit sampler seeds and a fixed Python hash seed, archive the pairs and predictions, and compute both definitions on these same pools. Thus the paired definition comparison is exact for the re-generated pools; differences from previously reported numbers can also include Monte-Carlo variation. Identity and equivalent-rewrite checks use $M = 10$, $N \leq 80$, and seed 0. The convergence sweeps use seeds $\{11, 22, 33, 44, 55\}$. Seed sensitivity uses $\{11, 22, 33, 44, 55, 66, 77, 88, 99, 1010\}$; other stability sweeps use $\{11, 22, 33\}$. Corruption uses five perturbation seeds $\{101, 202, 303, 404, 505\}$, $M = 8$, and sampler seed 17. Label noise uses $M = 10$ and seeds $\{11, 22, 33\}$; imbalance uses $M = 8$ and those same three seeds. The candidate for stability, noise, and imbalance is generated with perturbation seed 7. Experiment D uses perturbation seeds 1000–1007 and sampler seeds 2024–2031, with $M = 8$. Except in explicit sweeps, $V = 3$ and $N \leq 80$. BoundaryAcc means retain the original order of averaging over seeds and rules; we do not pool all pairs across rules into a new denominator. The same regenerated BC pools are reused across label-noise and class-ratio levels. Figures and tables use the same canonical easy-split runs as the all-splits summary. accuracy-tied groups are always formed within each GT rule. The controlled experiments use CPU computation only, with no model training, GPU computation, or external model API.

A.4 Data and Method Configurations

A.5 Data Sources

PROSITE. PROSITE signatures describe conserved protein sequence features that are associated with protein families, domains, or functional sites. These tasks include short motifs, typically within 20 amino acids, described by PROSITE patterns, as well as longer-range features described by position-specific scoring models. Each signature has a curated name and is accompanied by reference literature on its PROSITE entry page. These signatures are derived from literature-based curation and computational analysis, and PROSITE reports their matches to the corresponding biological sites.

JASPAR transcription factor binding sites. Each task corresponds to a transcription factor motif, where positive examples are extracted from annotated binding-site sequences together with their genomic coordinates. Such motif features are often represented by position frequency matrices (PFMs). We used JASPAR records from six species and their corresponding reference genomes: *Arabidopsis thaliana* (araTha1), *Caenorhabditis elegans* (ce11), *Drosophila melanogaster* (dm6 and dm3), *Mus musculus* (mm10), *Homo sapiens* (hg38), and *Saccharomyces cerevisiae* (sacCer3).

Table 14: Dataset statistics by source and domain.

Source	Domain	#Tasks	Sample Count	Pos./Neg. Ratio	Seq. Length	Avg. Length
Real	AMP	3	1,334–12,696	0.5006–1.0654	1–190	27.80
	Gene	13	2,119–223,854	0.2000–1.0219	40–70	45.07
	MA	47	300–7,926	1.0000–1.0000	9–20	12.19
	PS	43	300–17,683	1.0000–1.7270	11–592	36.84
	SP	3	21,311–21,311	0.0161–0.1414	30–70	69.91
Virtual	Virtual1	50	100,997–102,000	0.0313–21.7888	10–27	16.01
	Virtual2	50	100,988–102,000	0.0316–21.0139	9–31	17.24
	Virtual3	50	100,827–102,000	0.0322–18.1942	9–31	17.19
	Virtual4	50	85,396–102,000	0.0344–18.4120	10–28	16.51
	Virtual5	50	44,581–101,991	0.0304–28.9183	8–26	16.12

Benchmark tasks for the DNA foundation-model. We mainly select binary classification tasks whose positive and negative examples are real sequences and whose sequence lengths are below 100 bp, including bacterial promoter recognition, short-window human promoter recognition, 4mC site recognition, and 5mC/6mA methylation-site recognition. These tasks involve either motif-like sequence features or exhibit known statistical conservation patterns.

Signal peptide tasks. Signal peptides are short N-terminal sequence regions that guide newly synthesized proteins to the secretory pathway or specific cellular compartments. Following the original signal peptide categorization, each task distinguishes one signal peptide type from the remaining sequence classes. We select three SignalP categories with clear structural features and sufficient sample sizes: `SignalP_LIPO_vs_others`, `SignalP_SP_vs_others`, and `SignalP_TAT_vs_others`. We use SignalP 6.0 as the training source and SignalP 5.0 as the test source.

Antimicrobial peptide tasks. For peptide-level protein tasks, we include antimicrobial peptide classification datasets, where the goal is to distinguish AMP sequences from non-AMP short peptides or proteins. The discriminative signal of AMP tasks is more likely to involve soft physicochemical regularities, such as enrichment of positively charged residues, hydrophobicity, amphiphilicity, aromatic residues, and local residue-composition preferences.

A.6 Data Statistics

Table 14 summarizes the statistics of the real and synthetic datasets. Synthetic datasets contain a larger proportion of negative examples because their negative construction is more diverse. They also focus on individual motif rules, where the entire sequence segment is generated to satisfy or violate a target rule. In contrast, real biological sequences may only be partially explained by motif-like features, since the relevant rule can appear in a local region rather than covering the full sequence. As a result, real sequences have a slightly longer and wider length range than virtual motif instances.

A.7 Method Configurations

Symbolic rules-discovery methods. The execution procedures for all baselines are summarized in Table 15. FlexFringe uses the EDSM algorithm with the default parameter settings; therefore, we do not report its specific hyperparameter settings separately. For LLM-SR, Hypothesis Refinement, and Self-Refinement, all three methods involve feedback-driven refinement but differ in how feedback is defined. Hypothesis Refinement provides concrete feedback based on misclassified examples, whereas Self-Refinement proposes a more general feedback framework for heterogeneous tasks. Therefore, we follow the overall refinement framework of Self-Refinement while using task-specific misclassification feedback for rule discovery.

LLM-SR was originally designed for symbolic regression, where candidate equations are externally evaluated using the mean squared error as the loss. In our setting, the output is a hard symbolic rule that produces binary predictions. Therefore, we replace the regression loss with a classification loss defined as $1 - \mathrm{Acc}$, which is equivalent to minimizing the error rate in the evaluated examples.

Table 15: Overview of symbolic rule discovery baselines.

Baseline	Method Workflow
Few-Shot LLMs	The LLM is promted to output a DSL rule based on few-shot examples.
Agentic Methods	The agentic baselines provide the model with an ephemeral workspace containing the full training set and a shell sandbox, allowing the agent to conduct its own statistical analyses before committing to a rule.
Hypothesis Refinement (Qiu et al., 2024)	LLM performs multi-round search: in the first round, it generates n candidate rules, selects the best-performing rule, and then conducts up to $\texttt{max_iter} - 1$ additional refinement rounds, where feedback from misclassified examples is used to revise the rule.
Self-Refinement (Madaan et al., 2023)	A multi-round LLM-based rule discovery method. The first round generates an initial rule. In each subsequent round, the feedback is generated based on misclassified training examples, and the rule is revised until the maximum number of iterations is reached or the model determines that no further refinement is needed.
LLM-SR (Shojaee et al., 2025)	It performs DSL program search through the LLM-SR pipeline.
FlexFringe (Verwer & Hammerschmidt, 2017)	FlexFringe constructs a prefix-tree acceptor from positive and negative sequence samples. The main adaptation is to map each character in the original sequence alphabet to a corresponding 0–1 encoding. FlexFringe uses its original output format rather than our DSL.
Symbolic Search	This baseline performs rule search using simulated annealing (SA) and genetic algorithm (GA). Each individual corresponds to an AST (Abstract Syntax Tree)-encoded rule, and the fitness function uses F1 score as the evaluation objective.

For all baselines, we follow the default parameter settings reported in their original papers. The key settings are summarized in Table 16.

Black-box predictive baselines. We consider two types of black-box references. For domain-specific pretrained models, we use DNABERT-2 for DNA tasks and ESM-2 for protein tasks. For domain-agnostic neural models, we use one-dimensional CNN and ResNet models.

DNABERT-2 is selected as the black-box DNA reference because it is a general-purpose pretrained model for genomic sequences. It adopts an encoder-only Transformer architecture trained with a masked language modeling objective and improves DNA sequence tokenization through byte-pair encoding.

ESM-2 is selected as the reference protein in the black-box because it is a strong general-purpose protein language model for amino-acid sequences. Although newer variants of ESM (Hayes et al., 2025) incorporate additional modalities and target broader settings for protein-generation, ESM-2 remains a suitable and competitive choice for learning sequence-level representation.

For both pretrained models, we use a two-stage embedding-and-classification pipeline. The tokenizer and model are loaded and then tokenize each sequence and perform forward inference. Token-level representations are then pooled into sequence-level embeddings The resulting train and test embeddings are saved and passed to a downstream classifier.

Because the benchmark contains tasks with substantially different dataset sizes, sequence lengths, alphabets, and class distributions, all black-box references use grid search to select hyperparameters on the training data. Unless otherwise specified, CNN and ResNet are trained for 10 epochs with a batch size of 64. For ESM-2 and DNABERT-2, the pretrained encoders are frozen and used only for embedding extraction, with batch size 64; only the downstream elastic-net logistic regression classifier is tuned. Table 17 summarizes the model/checkpoint, pooling, and classifier settings, and Table 18 reports the grid search spaces and selection criteria.

A.8 Prompt Templates

A.8.1 Prompt for Rule Generation

For synthetic data generation, rule specifications are generated separately for each sequence domain. We present the prompt template using the DNA domain as an example.

Table 16: Key parameter settings of symbolic rule-discovery baselines.

Method	Main Setting	Candidate Rules	Temperature
Few-Shot LLM	Generate one rule	1 rule per run	0.025
LLM-SR	2500 iterations / samples	Iterative symbolic search	0.8
GA	Pop. 50; 1000 iterations; crossover/mutation 0.8/0.1; elite 5	$\leq$10000 rules; 3600 s	–
SA	Temp. 1000.0$\rightarrow$0.1; exp. cooling 0.95; 10 iter./temp.; 10 neighbors	$\leq$10000 rules; 3600 s	–
Hypothesis Refinement	3 refinement iterations	5 candidates per iteration	0.7
Self-Refinement	3 refinement iterations	1 refined rule per run	0.7
Codex	`codex exec`; reasoning effort set to high; 900 s wall-clock; six-script soft budget	–	–
Claude Code	`claude --print`; Bash and Write tools only; six analysis scripts; 1800 s wall-clock; $5 budget per rule	–	–

Table 17: Black-box method settings.

Method	Domain	Checkpoint	Params	Pooling	Classifier
CNN	all	–	config-dependent	–	binary classification head
ResNet	all	–	config-dependent	–	binary classification head
ESM-2	protein	`esm2_t36_3B_UR50D.pt`	3B	mean	elastic-net logistic regression
DNABERT-2	dna	`DNABERT-2-117M`	117M	mean	elastic-net logistic regression

DNA Rule Generation Prompt Template

System Role
The model is instructed to act as an expert in molecular biology and genomics, with a focus on DNA sequence motif analysis and gene regulation.

Task Description
1. The task is to generate {`num_rules`} diverse and biologically realistic DNA motif rules.
2. Each generated rule must describe a plausible genomic regulatory element, such as a promoter element, transcription factor binding site, CpG-rich element, enhancer-related motif, or palindromic restriction-like site.
3. The rules must be expressed in the specified DSL grammar.
4. Each rule contains a required symbolic pattern and optional logical constraints.
5. The generated rules should be diverse in sequence length, GC/AT composition, symmetry structure, and functional context.
6. The output must contain only the generated rules, without explanations or markdown.

Domain Knowledge
The prompt provides DNA-specific biological background, including:
1. DNA alphabet and valid symbols from the domain configuration.
2. Watson–Crick base-pairing relations: `A`–`T` and `G`–`C`.
3. Biological properties of base pairs, including the stronger hydrogen bonding of GC pairs compared with AT pairs.
4. Representative DNA motifs used as inspiration, including TATA boxes, CAAT boxes, GC boxes, E-boxes, CpG islands, Kozak sequences, palindromic restriction sites, and polyadenylation signals.

Complexity Control
The prompt specifies a target complexity level through:

```
{complexity}
{complexity_guide}
```

This controls the expected rule structure, including motif length, the use of conserved positions, wildcard regions, and whether logical constraints are needed.

DSL Grammar
Each rule must follow the provided grammar:

```
{grammar}
```

The pattern syntax supports:
1. Literal DNA symbols, such as `A`, `T`, `G`, and `C`.
2. Single-position wildcard: `x`.
3. Fixed-length wildcard region: `x(k)`.

Table 18: Grid search settings for black-box baselines.

Method	Tuned component	Grid search space	Selection
CNN	1D CNN classifier	$lr \in \{10^{-3}, 5\times10^{-4}\}$; $channels \in \{64, 128\}$; $kernel_size \in \{3, 5\}$; $dropout \in \{0.1, 0.3\}$	validation F1
ResNet	1D ResNet classifier	$lr \in \{10^{-3}, 5\times10^{-4}\}$; $channels \in \{64, 128\}$; $kernel_size \in \{3, 5\}$; $dropout \in \{0.1, 0.3\}$	validation F1
ESM-2	Elastic-net LR	$C \in \{0.1, 1, 10\}$; $l1_ratio \in \{0.1, 0.5, 0.9\}$; $class_weight \in \{\texttt{balanced}, \texttt{None}\}$; $max_iter = 1000$; $tol = 10^{-4}$	4-fold CV, ROC-AUC
DNABERT-2	Elastic-net LR	$C \in \{0.1, 1, 10\}$; $l1_ratio \in \{0.1, 0.5, 0.9\}$; $class_weight \in \{\texttt{balanced}, \texttt{None}\}$; $max_iter = 1000$; $tol = 10^{-4}$	4-fold CV, ROC-AUC

4. Variable-length wildcard region: `x(m,n)`.
5. Allowed character classes, such as `[AG]`.
6. Excluded character classes, such as {`AT`}.
7. Hyphen-separated pattern tokens ending with a period.

Logic Constraint Principle

A central instruction is that logical constraints should express properties that cannot be directly encoded by the pattern.

1. Constraints should not repeat information already fixed in the pattern.
2. Position-relation constraints may express equality, inequality, or DNA complementarity between conserved positions.
3. Conditional constraints may express dependencies between positions.
4. Global count or GC-content constraints should only be used when wildcard regions leave the composition underspecified.
5. Redundant constraints, such as adding a GC-content constraint to a pattern that is already fully GC, are explicitly discouraged.

Examples of valid non-redundant constraints include:

```
SAME_SYMBOL(pos1,pos6)
COMPLEMENT_SYMBOL(pos2,pos5)
MIN_SYMBOL_COUNT(G,5)
MAX_GC_CONTENT(0.4)
IF_POSITION_IS(pos1,A) THEN_POSITION_IS(pos3,T)
```

DNA-Specific Rule Design

The prompt encourages several biologically meaningful rule types:

1. Palindromic or reverse-complement-like motifs using complement constraints.
2. Symmetric transcription-factor binding motifs using equality constraints.
3. AT-rich promoter-like regions using AT-biased patterns or composition constraints.
4. GC-rich regulatory regions or CpG-like motifs using GC-content and symbol-count constraints.
5. Flexible motifs with conserved cores and bounded wildcard regions.

Length Constraints

The prompt imposes hard sequence-length constraints:

1. Minimum length: 10 bp.
2. Maximum length: 50 bp.
3. Typical length: 15–35 bp.
4. Simple rules are expected to be around 10–20 bp.
5. Medium rules are expected to be around 15–35 bp.
6. Complex rules are expected to be around 25–45 bp.
7. Very short patterns, overly long wildcard regions, and patterns exceeding 50 bp are rejected.

For example:

```
T-A-T-A-x(6,12)-A-T.
G-C-x(8,15)-G-C-x(8,15)-C-G.
[AT]-x(5,10)-[GC]-x(5,10)-A.
```

Quality Checks

Before outputting each rule, the model is instructed to verify:

1. The domain is DNA.
2. Only DNA symbols `A`, `T`, `G`, and `C` are used.

3. Each generated pattern is unique.
4. The total sequence length is within 10–50 bp.
5. All position references in logical constraints are valid.
6. The rule is biologically plausible.
7. GC content and composition constraints are reasonable.
8. Logical constraints are non-redundant with the pattern.

Output Requirements
1. Generate exactly {num_rules} different DNA motif rules.
2. Each rule must follow the specified BNF/DSL format.
3. Each rule must be unique and syntactically valid.
4. The output must contain only the rules.
5. No natural-language explanation, markdown, or extra text should be included.

Generation Instruction

```
OUTPUT {num_rules} DNA MOTIF RULES NOW

CRITICAL:
1. Count conserved bases before writing logic.
2. No duplicate patterns.
3. DOMAIN must be DNA.
4. Position references must be valid.
5. Total sequence length must be 10--50 bp.
```

A.8.2 Prompt for Rule Discovery

For rule discovery, we construct a domain-parameterized prompt template. Given a domain, the template selects the corresponding alphabet, macros, and representative rule examples, while keeping the pattern syntax, logic-constraint syntax, task objective, and output format consistent across tasks.

Rule Discovery Prompt Template

System Prompt
You are performing a symbolic sequence rule discovery task. Given labeled positive and negative sequences from a scientific or synthetic sequence domain, discover an executable and interpretable symbolic rule that distinguishes positive examples from negative examples.

Task Description
1. You are given a set of positive and negative sequence examples.
2. Positive examples satisfy an unknown target rule, while negative examples do not.
3. Your goal is to discover a concise symbolic rule that explains the positive examples and rejects the negative examples.
4. The discovered rule must be written in the provided DSL format.
5. The rule should consist of a core sequence pattern and, when necessary, additional logical constraints.
6. The rule must be executable by the DSL parser and matcher.
7. Avoid memorizing individual sequences. Prefer simple, general, and biologically or structurally plausible rules.
8. Return exactly one DSL rule block and no extra commentary.

Domain Configuration
1. Domain: {domain}
2. Valid alphabet:
   {alphabet}
3. Domain-specific property classes:
   {property_classes}
4. Complement relations, if applicable:
   {complement_relations}

Only use symbols and property classes that are valid for the current domain.

DSL Rule Format
The output rule must follow this format:

```
RULE_ID: AUTO
```

```
DESCRIPTION: <brief natural-language description>
DOMAIN: <domain>
PATTERN: <pattern>.
LOGIC_CONSTRAINTS:
<constraint_1>
<constraint_2>
...
```

If no logical constraint is needed, write:

```
LOGIC_CONSTRAINTS:
NONE
```

Pattern Grammar

The `PATTERN` field describes the local sequence structure. The pattern must end with a period.

1. Literal symbol: `A`, `C`, `G`, etc.
2. Wildcard: `x`
3. Fixed-length gap: `x(k)`
4. Variable-length gap: `x(m,n)`
5. Allowed symbol set: `[ABC]`
6. Excluded symbol set: {`ABC`}
7. Pattern tokens must be separated by hyphens.
8. Do not use unsupported regex operators such as `*`, `+`, `?`, `|`, or grouping parentheses.

Valid pattern examples:

```
A-C-D-x-K.
C-x(2)-[DE]-x(1,3)-K.
[AG]-x-{P}-T.
x(1,4)-G-[ST]-x.
```

Logical Constraint Grammar

Use logical constraints only when the positive and negative examples cannot be explained well by the pattern alone.

Available constraints include:

1. `SAME_SYMBOL(pos1,pos2)`
2. `DIFFERENT_SYMBOL(pos1,pos2)`
3. `COMPLEMENT_SYMBOL(pos1,pos2)`
4. `MIN_SYMBOL_COUNT(symbol,k)`
5. `MAX_SYMBOL_COUNT(symbol,k)`
6. `IF_POSITION_IS(pos,symbol) THEN_POSITION_IS(pos,symbol)`

Position convention:

1. Positions in logical constraints refer to conserved pattern positions.
2. Wildcards and gap tokens do not introduce named positions unless explicitly matched as concrete conserved tokens.
3. Use position indices consistently with the pattern.

Discovery Strategy

When discovering the rule, follow these priorities:

1. Identify conserved symbols or short motifs that appear frequently in positive examples.
2. Compare positives against negatives to avoid selecting motifs that also appear in negatives.
3. Use character classes when several symbols can appear at the same position.
4. Use fixed or variable gaps when the conserved symbols occur with flexible spacing.
5. Add logical constraints only if they explain systematic dependencies that cannot be captured by the pattern alone.
6. Prefer the simplest rule that separates positives from negatives.

Few-shot Example 1: Pattern-only Protein Rule

Positive examples:

```
ACDEK
ACDFK
ACDGK
```

Negative examples:

```
ACDPP
GGDEK
ACDAA
```

Answer:

```
RULE_ID: AUTO
DESCRIPTION:
Protein sequences with a conserved A-C-D prefix and K suffix.
DOMAIN: PROTEIN
PATTERN: A-C-D-x-K.
LOGIC_CONSTRAINTS:
NONE
```

Few-shot Example 2: DNA Rule with Logical Constraints
Positive examples:

```
ATAT
CGCG
TATA
GCGC
```

Negative examples:

```
AAAA
ATGG
CCGT
GCAA
```

Answer:

```
RULE_ID: AUTO
DESCRIPTION:
DNA sequences whose paired positions satisfy complementarity.
DOMAIN: DNA
PATTERN: x-x-x-x.
LOGIC_CONSTRAINTS:
COMPLEMENT_SYMBOL(1,4)
COMPLEMENT_SYMBOL(2,3)
```

Input Examples
Positive examples:

```
{positive_examples}
```

Negative examples:

```
{negative_examples}
```

Output Requirements
1. Output exactly one DSL rule block.
2. Do not output explanations, analysis, markdown, or Python code.
3. Do not output multiple candidate rules.
4. Do not use symbols outside the domain alphabet.
5. Do not use unsupported DSL syntax.
6. The final answer must be directly parseable as a DSL rule.

Return the discovered DSL rule now.

Prompt adaptation for LLM-based baselines. All LLM- and agent-based baselines use the same DSL specification, input and output format, but the prompt is adapted according to each method's pipeline.

Few-shot LLM prompts place labeled training examples in the context, and the model is asked to output a complete DSL rule.

In LLMSR, the model only proposes candidate rule skeletons. The prompt does not contain training examples.

In hypothesis refinement, the model starts from an initial rule and revises it through failure-case feedback. At each iteration, misclassified examples are cached as feedback records in the following format:

$$\text{Sequence: } \{\text{sequence}\}, \quad \text{Expected label: } \{\text{target}\}, \quad \text{Actual label: } \{\text{pred}\}.$$

The current rule and all cached feedback records are then passed to the model for revision. If no misclassified example remains, the refinement process is terminated.

In self-refinement, the model critiques and revises the current executable DSL rule. The feedback prompt contains the current rule and the most recent misclassified examples. The critique is required to analyze pattern errors and logic-constraint errors separately. Pattern errors include overly broad or overly narrow patterns, wrong anchors, wrong token sets, and wrong gap lengths. Logic-constraint errors include missing constraints, unnecessary constraints, wrong symbol-count constraints, and unsupported constraint ideas. The model outputs `STATUS: PASS` or `STATUS: FAIL` with several concrete revision suggestions. If the model outputs `STATUS: PASS`, the iteration stops; otherwise, the revised rule is evaluated in the next round.

For the agent-based methods, the prompt specifies the task objective, the domain alphabet (with complement relations where applicable), the DSL grammar (the full BNF together with an abbreviated cheat sheet), a small set of reference rules included purely as format examples, the available tools, and the rule-output contract. The training set is not embedded in the context: the agent is instead given a workspace containing a single `train.txt` file (one row per labelled sequence) and is instructed to author its own Python analysis scripts—via the `Write` tool for Claude Code and shell heredocs for Codex—run them, and read the printed statistics to identify the discriminative motif. The prompt enforces a tight budget of at most six analysis scripts and twelve tool calls per rule, together with explicit early-stopping criteria, after which the agent must commit to its current best rule and emit the final `RULE_ID` block.

This may improve transparency in biological sequence analysis and support hypothesis generation for downstream scientific validation.

A.9 Limitation and Future Work

SeqRule-Bench defines rules based on a DSL grammar. This setting restricts the rule space, making fair and systematic evaluation feasible. It reflects our emphasis on human interpretability and is consistent with the intended trade-off in this work. The grammar is grounded in PROSITE, providing a biological basis for its design. However, this approach may introduce a representation bias, as rules conforming to the grammar are easier to validate, whereas meaningful rules outside it may be penalized. Future work could explore a richer variety of grammatical forms to accommodate a broader range of biological tasks.