CODING LLMS CAN BE HARNESSED TO LEARN INTERPRETABLE MODELS FROM DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Since compact, interpretable classifiers correspond to simple programs, it is natural to ask if LLM-based coding agents can be used to learn interpretable models from data. In this paper we present a simple harness that repeatedly prompts a coding LLM to produce the next rule in a rule set, given a sample of labeled examples. We show that this approach is competitive with, and often superior to, state-of-the-art rule-learning methods on tabular data, especially with small training sets, and wins by large margins on text, including against another LLM-based rule learner; that this approach outperforms standard harnesses, and that the approach immediately leads to new kinds of interpretable models, since coding agents are not restricted to search a highly restricted hypothesis space. In addition to the advances made in learning interpretable models, these results are of interest because they introduce a new class of coding benchmarks—construction of interpretable classifiers—which includes quantitative measurements of output quality (i.e., test-set error) and existing mature "specialist coding algorithms" (i.e., existing learning methods).

1 INTRODUCTION

Interpretable symbolic models (e.g., rules and decision trees) are a useful alternative to black-box models in some practically important settings; however, learning interpretable classifiers remains a challenging task (Rudin et al., 2022). Since learned interpretable models correspond to short programs, the development of LLM-based coding agents raises a natural question: **how well can LLM-based coders produce accurate interpretable symbolic models?**

Past work (Singh et al., 2023; Han et al., 2024; Abhyankar et al., 2026; Zhang et al., 2025; Jia et al., 2026) shows that LLMs can use their background knowledge to generate *features* for symbolic learners: e.g., FeatLLM uses ensembles of prompted LLMs to generate hundreds of binary features predicting each class, which are then used by a downstream classifier learner. Much past work on LLM-based feature generation focuses on few-shot learning settings, like FeatLLM, or NLP problems, like Aug-imodels (Singh et al., 2023). We consider here the more difficult task of **building complete, fully executable, interpretable classifiers with LLMs**, including tabular classifiers in many-shot settings. There is little prior work on this task, although there has been work on a zero-shot variant of the task (Carrasco et al., 2025) and at least one concurrent work (Kovács et al., 2026): see Section 2.

Of course, practitioners often do use coding LLMs to build (or help build) interpretable models: e.g., in investigating the recent OpenAI/HuggingFace hacking incident, safety researchers "iterated with AI agents to design simple regexes to classify messages" (METR, 2026). Of course, the same AI agents could have been used to directly classify messages instead, but this was undesirable—elsewhere in the same report LLM agents were observed to be biased, e.g., to "present an overly charitable picture of agents' reasoning and deceptive behaviors" (METR, 2026). Although we will compare where possible to interpretable models jointly built by humans and LLMs, we focus here on **fully autonomous systems for building interpretable classifiers from data**.

LLMs have obvious practical advantages over traditional symbolic learners for generating classifiers, just as they do for feature generation: they can exploit background knowledge, and they can use programming language constructs that do not lend themselves to discrete search (like regular ex-

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.

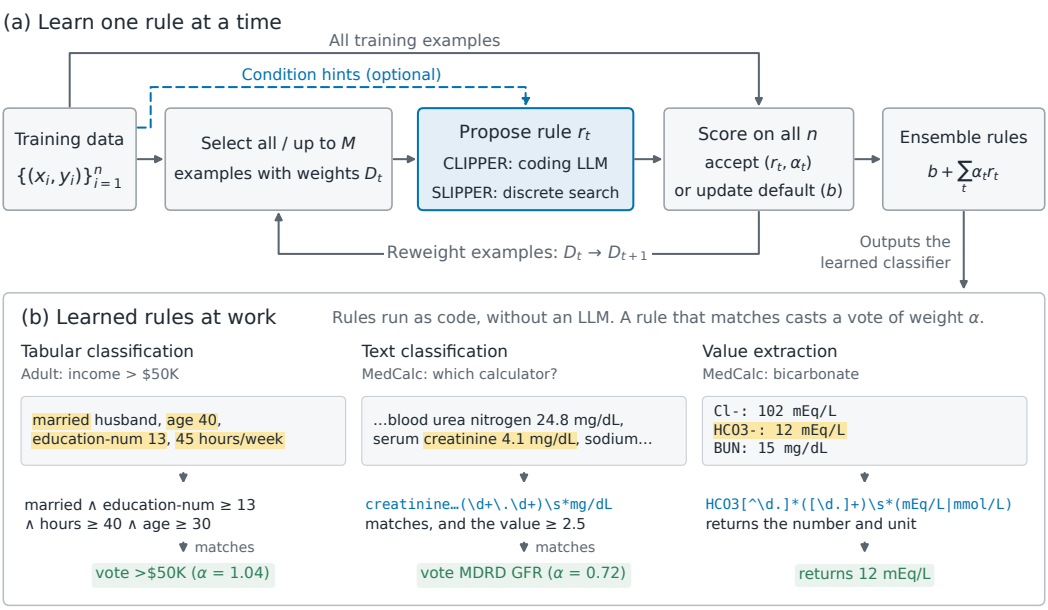


Figure 1: Learning a rule classifier with CLIPPER. (a) CLIPPER replaces SLIPPER's discrete rule search with a coding LLM. The harness iteratively samples the training set using the current distribution; sends the sample to the LLM, prompting it to produce a rule; scores the proposed rule on the full train set; either accepts a rule or updates the default; and reweights the train examples for the next round. (b) Summary of three actual CLIPPER-learned rules: attribute tests correlated with high income; a regex that finds a high creatinine value, correlated with the MDRD GFR calculator; and a regex that extracts a bicarbonate value and corresponding units. Yellow highlights mark relevant parts of the input; blue text is copied from a learned rule. Note that no LLM is called at inference.

pressions). In addition to the practical value of improving the performance of interpretable learning methods, another motivation for this work is that **writing predictive rules is different from existing programming benchmarks** like bug-fixes or competitive-programming puzzles. Test-set error provides a quantitative measure of quality, but not an indication of when a classifier is "finished"; qualitatively, many LLMs have a tendency to "overthink" on this task, generating long, expensive, and unproductive outputs. There is also a **strong existing baseline**, since mature symbolic learners are competent "specialist coders" for this task.

Our approach is a conservative modification of a classic rule-learning method. A harness implements the learner's outer loop: in each iteration, it reweights the training data to focus on examples that are not explained by the accumulated rules, and then submits a sample of them to an LLM agent, prompting it to construct one rule based on the sample. Specifically, the harness follows the outer loop of SLIPPER (Cohen & Singer, 1999), which uses a confidence-rated boosting procedure. This enables a direct comparison of LLM coding agents to SLIPPER's discrete rule search[1] to construct a rule as its inner loop.

We make the following contributions in the paper.

A new rule learner. Experiments with 12 interpretable models on a suite of eight tabular datasets often studied in the interpretability literature (Kruschel et al., 2026) show that SLIPPER is a strong baseline (see Figure 2). Building on this we propose an LLM-based rule learner called CLIPPER (Coding LLMs used for Iterative Prompting to Produce Error Reduction). On tabular datasets, CLIPPER outperforms existing rule learners with smaller training sets, and is statistically indistinguishable from the strongest baseline given appropriate tools and "hints" (relevant statistics that are added to the data sample in the prompt). On text datasets, CLIPPER wins by large margins over similar baselines, including RuleChef (Kovács et al., 2026), another LLM-based rule learner. CLIPPER can even be extended to use regular expressions that perform extraction, and in a comparative study,

[1]Which is based on the search used in RIPPER (Cohen, 1995).

CLIPPER (using Gemini 3.1 Flash-Lite) found extractors comparable to models constructed by working interactively with a frontier coding agent (Fable 5.1).

Analysis and technical innovations. Controlled comparisons show that CLIPPER's coding agent consistently outperforms SLIPPER's discrete search given the same information, and additional experiments support the claim that *background knowledge* is the reason. We also show that the same harness works comparably for multiple model families. We compare to general-purpose coding harnesses, and show that the CLIPPER harness outperforms them on our test suite.

2 Related work

Symbolic classifiers. We build on, and compare to, prior symbolic learners, in particular rule learners. Our work is most closely related to RIPPER, which uses an effective but efficient search for a compact rule set (Cohen, 1995), and SLIPPER, which uses confidence-rated boosting (Cohen & Singer, 1999; Schapire & Singer, 1999) to build a comparably small ensemble of weighted rules. We use our own re-implementation of SLIPPER[2] and CLIPPER shares most of the code, but replaces the discrete rule search with a call to a Pydantic AI[3] agent. We compare to other models, including CORELS, which searches for an optimal regularized rule list over a fixed set of antecedents (Angelino et al., 2018); RuleFit, which is a hybrid of rule models and linear classifiers (Friedman & Popescu, 2008); FIGS, which learns an ensemble of small trees (Tan et al., 2025); and others. Many of these comparisons were facilitated by the `imodels` package (Singh et al., 2021).

LLM-generated features for downstream symbolic learners. Past work (Singh et al., 2023; Han et al., 2024; Abhyankar et al., 2026; Zhang et al., 2025; Jia et al., 2026) uses LLMs to augment or select features for symbolic learners. One well-known example is FeatLLM (Han et al., 2024), which uses bagging and prompting in a few-shot setting to generate many conjunctive rules (much like the ones produced by SLIPPER), which are used by a symbolic learner. CLIPPER differs in generating a compact classifier directly, similarly to SLIPPER: e.g., in most of the experiments from Han et al. (2024), 200 rules are generated for each class, from up to 64 examples, while CLIPPER generates an average of 9.6 rules on 2000 examples, and SLIPPER generates an average of 8.9.

LLMs for interpretable model construction. A zero-shot decision tree learner is described by Carrasco et al. (2025), where the learner is modified so that data-derived statistics are replaced with LLM-driven decisions. Our work is similar in spirit but combines LLM background knowledge with data. In concurrent work, RuleChef describes an agentic system that generates executable rules for text classification and extraction (Kovács et al., 2026). We compare to it in Section 4.2 and Appendix A.6. Unlike RuleChef, CLIPPER is evaluated on tabular data, and our framework allows controlled evaluation of an LLM's ability to produce rules from data.

Boosting language-model classifiers. Manikandan et al. (2023) use LLMs as weak learners on tabular data: each round produces a natural-language summary, which conditions an LLM when predicting labels. CLIPPER also uses boosting to combine weak learners, but each weak hypothesis is executable code, so prediction needs no LLM.

3 Method

Before deciding to develop CLIPPER we evaluated 12 interpretable learners on 8 tabular benchmarks previously selected by other authors (Kruschel et al., 2026) as being appropriate tasks for use of interpretable models (e.g., COMPAS estimates the likelihood of recidivism). We constructed binary versions of each of these datasets, and evaluated error rates at up to five training-set sizes.[4] Running 10 seeds for each method gives 4510 experiments, shown in Figure 2, and SLIPPER emerged as the strongest baseline learner, followed by CORELS.

SLIPPER's learning method lends itself to adaptation to a harness, which we describe as follows, letting $(x_i, y_i)_{i=1}^n$ be a binary training set with $y_i \in \{-1, +1\}$, and letting $D_t(i)$ be the normalized weight of example i at round t. For $t = 1$, D_t is uniform.

[2] A Python rewrite of the original C implementation.

[3] `https://pydantic.dev/`

[4] *college*, with only 750 training instances, uses only 3 sizes.

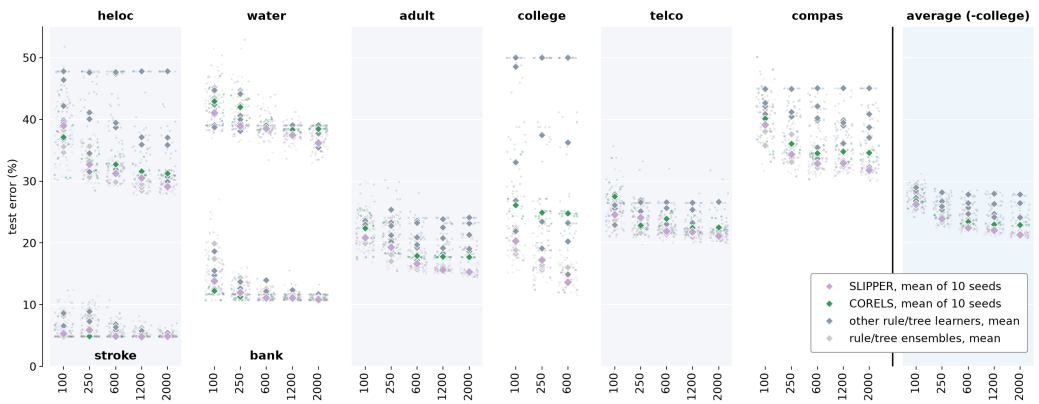


Figure 2: **SLIPPER is a strong symbolic baseline.** On 4,510 experiments, SLIPPER (purple) and CORELS (green) are the two strongest rule learners. Weaker rule and tree learners are blue; ensembles are gray. The shaded band (right) is average performance over the 7 larger benchmarks. In the first 6 bands the faint dots are individual runs with different seeds (jittered to reduce overlap) and the diamonds are means; in the shaded band each dot is one seed's average over the 7 benchmarks. Benchmarks with distinct error ranges are stacked to save space.

Finding a rule. In round t SLIPPER uses a greedy discrete search to find a "rule" r_t, i.e., a learned predicate $r_t(x) \in \{0, 1\}$ that "covers" an example when it returns one. The rule's positive coverage p_t, negative coverage q_t, and weight α_t are computed as

$$p_t = \sum_{i:y_i=+1} D_t(i) r_t(x_i), \quad q_t = \sum_{i:y_i=-1} D_t(i) r_t(x_i), \quad \epsilon = \frac{1}{2n}, \quad \alpha_t = \tfrac{1}{2} \log \frac{p_t + \epsilon}{q_t + \epsilon}. \tag{1}$$

Reweighting the examples. The rule weight α_t is used to update the example weights:

$$D_{t+1}(i) \propto D_t(i) \exp[-y_i \alpha_t r_t(x_i)] \tag{2}$$

Rejecting rules and the default rule. On round 1, SLIPPER defines the "default rule" $r_1(x) \equiv -1$ (for any x). The other rules always return 0 or 1, so this is the only rule that recommends the negative class. In subsequent rounds, SLIPPER's search tries (roughly speaking) to maximize $(\sqrt{p_t} - \sqrt{q_t})$ without overfitting. If this quantity is negative then r_t is replaced with r_1 and its weight α_1 is incremented by the appropriate α_t. The final classifier combines the rule weights and predicts the sign of $F(x)$:

$$F(x) = \sum_t \alpha_t r_t(x) = b + \sum_{t \in \mathcal{A}} \alpha_t r_t(x)$$

where b is a bias term (the contribution of r_1) and $\mathcal{A}$ indexes accepted rules.

SLIPPER's rules are sparse. Importantly $F(x)$ **only depends on the rules that cover** x, unlike the case for ordinary boosting, where all weak hypotheses contribute to every prediction.

Boosting as a harness and hyperparameters. **CLIPPER uses this same procedure, except that it invokes an LLM agent to build a rule.** One hyperparameter of SLIPPER (and thus CLIPPER) is the number of rounds of boosting T.[5] SLIPPER bounds $T < 100$, and uses 5-fold internal cross-validation to pick the best value T^*. CLIPPER is similar, but to reduce LLM usage it cross-validates SLIPPER (not itself) to pick T^*: the only exception is that because CLIPPER's rule builder is more productive, it always allows at least two rounds of boosting, even when SLIPPER's cross-validation prefers 0 or 1. **All other hyperparameters for CLIPPER were set on an independent tuning set of benchmarks.** That includes the prompts, the tools, and other configuration choices.[6] Hence in

[5] T upper-bounds the number of rules learned—the number of rules will be less than T if rules are rejected.

[6] We developed CLIPPER on a disjoint suite of ten binary tasks: Abalone, Breast cancer Wisconsin, Cylinder bands, German credit, Hypothyroid, Move1, Obesity, Pima diabetes, Portuguese student performance, and Titanic (Appendix C).

the results of Section 4, **every reported result measures performance on tasks that played no part in designing the method.**

Appendix A gives the full procedure.

Details of the prompting and the tools. The agent receives a weighted sample of at most M examples, drawn without replacement from D_t, together with labels, sampled weights, and the tabular data schema. For tabular data, we use $M = 100$ and a Markdown table with quoted string values to encode the data sample. Note that a different sample is passed to the LLM in each boosting round, so potentially many more than M can be seen by CLIPPER during learning. We also modified SLIPPER to use samples of size M in order to perform controlled experiments (below). The prompt asks for one conjunctive rule implemented as a Python function `covers(x)`, where x is an attribute dictionary. This restriction comes from prompting: the executor accepts more general Python predicates.

Variants. We compare four main CLIPPER variants. *Sample* provides the examples and schema alone. *Coverage* adds a tool that, given a predicate, returns values of p_t and q_t computed on the full training set. *Hints* adds to the prompt a table of the best single condition for each attribute, as found by SLIPPER's discrete optimizer, with sample and full-data coverage. *Hints+tools* also adds this hint output to the output of the coverage tool. Coverage uses a six-call tool limit per attempt; hints+tools uses two. All conditions receive the same full-data scoring by the harness after a rule is found.

Textual data and multiple classes. On tabular data, SLIPPER is restricted to rules containing conjunctions of equality tests for discrete attributes and inequality tests for continuous ones; it also can construct tests on the presence of sparse attributes (e.g., words, for classifying text). Using LLM rule builders allows generation of many other types of rules by slightly varying the prompt. We consider string operations and regular expressions for classification. We also consider rules that extract data from text. For extraction, a rule returns a value or abstains. In Equation 1, p_t becomes the weight of correct returned values and q_t the weight of incorrect returns; abstentions contribute to neither. We follow the same weight update procedure as in classification; the default is abstention, and a rule is retained only if $p_t > q_t$. At prediction time, confidences are summed for each canonical value and the largest sum wins. If no rule matches, the system abstains. We compare unrestricted Python extraction functions with structured regex rules containing patterns, flags, and a capture-group template.

For multiclass problems we use one-versus-rest prediction (Section 4.2).

4 EXPERIMENTS

We compare the coding agent with discrete rule search at equal information, extend it to more data, compare it with established and LLM-based rule learners on tabular data and text, and separate the roles of harness and model.

4.1 EVALUATION PROTOCOL

The tabular evaluation uses eight binary tasks drawn from the suite studied by Kruschel et al. (2026): Adult, Bank marketing, College, COMPAS, HELOC, Stroke, Telco churn, and Water potability. Each has a fixed, stratified 25% test holdout. For each training size and seed, methods receive the same training subset. As noted above, CLIPPER and SLIPPER use the same boosting budget (except CLIPPER always boosts at least twice). The default coding model is Gemini 3.1 Flash-Lite, a fast, inexpensive model, with default generation hyperparameters.

Results on tabular datasets report the mean test-error percentage and standard error of the mean (SEM) across 10 seeds (unless noted otherwise). Macro error averages dataset means equally, with propagated SEM. Learning curves that extend past $n = 750$ exclude college, which has only 750 training examples. Appendix A specifies additional details.

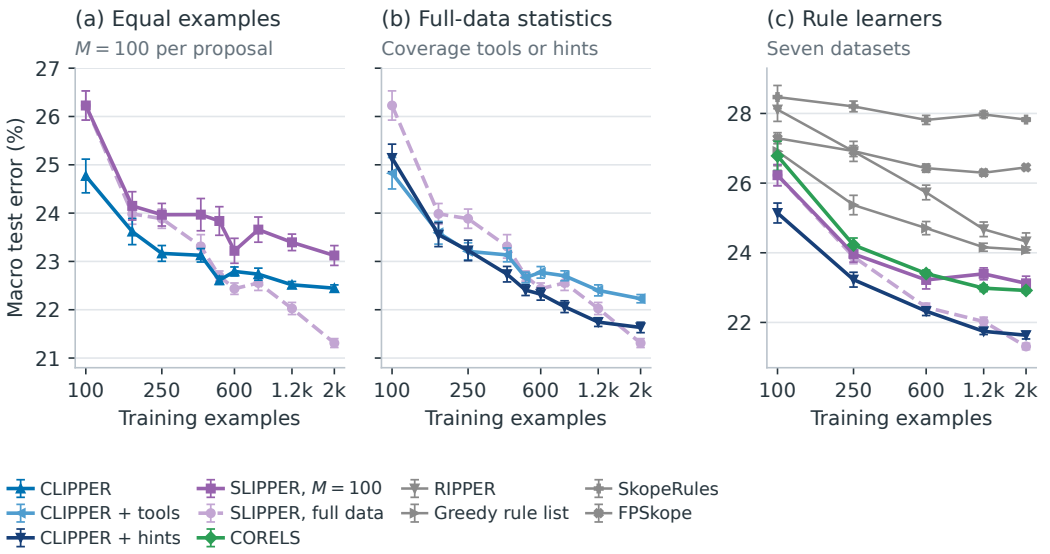


Figure 3: **CLIPPER on tabular data.** Macro test error on seven tabular datasets (mean and propagated SEM, ten seeds). CLIPPER sees at most $M = 100$ examples per proposal, and every proposal is scored on the full training set. (a) CLIPPER against SLIPPER given the same examples ($M = 100$) and given all training rows. (b) CLIPPER given full-data statistics in two ways: coverage tools or condition hints. (c) CLIPPER with hints against established rule learners.

4.2 LLMs BUILD BETTER RULES THAN DISCRETE SEARCH

With both rule builders limited to a sample of $M = 100$ examples drawn in each boosting round, CLIPPER has lower macro test error than SLIPPER at all nine training sizes (Figure 3a); at $n = 1{,}200$, CLIPPER's error is 0.9 ± 0.2 points lower (22.5 versus 23.4%). In this controlled experiment, the only difference is the way that rules are built: by statistics and discrete search for SLIPPER, and by LLMs for CLIPPER.

Of course, SLIPPER's rule-building does not need to limit itself to a sample of 100, as the dashed line in Figure 3a shows. One might hope that giving the LLM a larger sample would help, but this is *not* true.[7] However, giving direct access to full-data statistics improves performance substantially, especially with more training data. The "hints" variant of CLIPPER keeps the same sample size, but adds to the prompt the best single condition for each attribute, as found by SLIPPER's discrete search. Without hints, CLIPPER improves over full-data SLIPPER up to about $n = 500$, about 5 times the sample size. With hints, CLIPPER beats full-data SLIPPER at eight of nine sizes (Figure 3b). Coverage queries alone give smaller gains (Figures 3b and 6).

Figure 3c compares CLIPPER with hints against the other rule learners from Figure 2. CLIPPER has lower error than CORELS, RIPPER, the greedy rule list, SkopeRules, FPSkope, and sampled SLIPPER at all five training sizes; only full-data SLIPPER is lower, and only at $n = 2{,}000$. CLIPPER's learned rulesets are also small: across CLIPPER conditions and training sizes, the median ruleset contains 4.5 to 13.5 rules, well within the range humans can inspect (Heider et al., 2023; Appendix F).

Text classification. We also constructed a benchmark suite with five binary text-classification tasks, two for spam detection (SMS spam, YouTube spam) and three for sentiment (Amazon, IMDb, and Yelp reviews; Appendix A.5). We use CLIPPER without tools or hints, with two prompt variants:

[7] On the development suite, increasing the prompt from $M = 100$ to 200 or 400 examples *raised* macro error slightly (by 0.3–0.4 points; Appendix C). On MedCalc (Section 4.2), which has very large examples, CLIPPER with $M = 20$ is more accurate than with $M = 100$ (Table 9). Development suite experiments suggest that the Gemini model we use as a base gets less accurate at estimating dataset statistics as the sample grows.

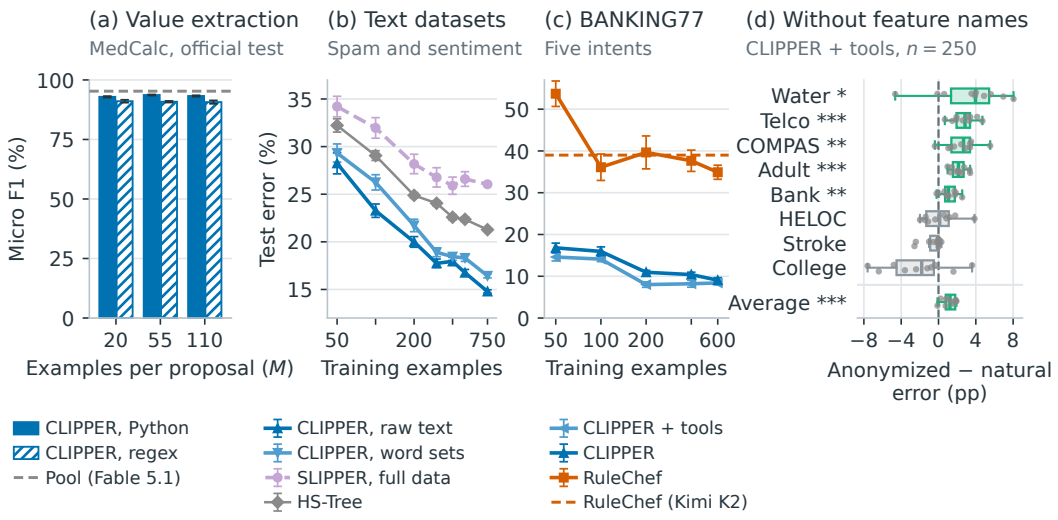


Figure 4: **CLIPPER on text.** Bars and points show means over ten seeds, with SEM error bars, propagated across datasets in (b); CLIPPER uses Gemini 3.1 Flash-Lite throughout. (a) Official-test value extraction (Section 4.3): micro F1 of Python and regex rules learned without coverage tools. Dashed: the reference pool, written with Claude Fable 5.1 and evaluated once. (b) Macro error on five text datasets. CLIPPER uses no coverage tools and reads either the raw text or, like SLIPPER, word sets. (c) Test error on five BANKING77 intents (200 official test queries). CLIPPER and RuleChef read the raw text with the same model and training queries. Dashed: RuleChef's reported error with Kimi K2 (Kovács et al., 2026), counting unlabeled queries as errors (our runs give them the most common intent). (d) Paired error changes when dataset, feature, class, and value names are replaced with arbitrary identifiers (all eight tabular datasets). Grey dots are seeds; Average is each seed's mean over the eight datasets. Boxes: quartiles, medians, 1.5-IQR whiskers. Stars mark paired t-test significance (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$); green boxes are significant increases.

one that directs the LLM to construct the same sort of rule conditions as are supported by SLIPPER ("word sets", which allow testing if a token appears) and one that allows regular expressions on the raw text. We compare to SLIPPER with the same conditions and no sampling. Here both variants of CLIPPER substantially improve on SLIPPER, and also on the best baseline learner (in this case HS-Tree, an `imodels` tree learner with hierarchical shrinkage, using TF-IDF-transformed features; Figure 4b). CLIPPER also does well on a 55-way task: predicting which MedCalc-Bench-Verified calculator (Khandekar et al., 2024) a patient note requires (Appendix B).

Recent systems also use an LLM to write rules. The closest to CLIPPER, RuleChef (Kovács et al., 2026), asks an LLM to write rules from labeled examples, but without boosting: it writes a small set of rules and then revises them to fix their mistakes. We compare the two on the benchmark RuleChef's authors use: five intents from BANKING77 (Casanueva et al., 2020), with all 200 official test queries for those intents (Appendix A.6). Both methods read the raw text and use the same model, Gemini 3.1 Flash-Lite, and RuleChef runs its authors' own benchmark script and settings. CLIPPER fits one-versus-rest models as for MedCalc, with $M = 100$. CLIPPER's error is lower than RuleChef's in all 50 runs (five training sizes, ten seeds each), by 29.7 points on average with coverage tools and 27.8 points without them (Figure 4c). From $n = 100$ on, RuleChef's error stays between 34.9 and 39.7%, while CLIPPER with tools reaches 8.0% at $n = 200$. RuleChef's rules label only about half of the test queries: after its first rounds of revision, it usually replaces its initial rules with one broad regular expression per intent.

Background knowledge. We believe that CLIPPER's ability to leverage its knowledge of language is why it has such strong performance on these text datasets. Reading the same word sets as full-data SLIPPER, CLIPPER without tools has lower error at every size (29.3 versus 34.2% at $n = 50$; 16.4 versus 26.0% at $n = 750$; Figure 4b). On a text dataset whose labels were published after the model's training cutoff, CLIPPER reading the raw text still has the lowest error at $n = 750$ (6.9%,

Table 1: **Iterative construction matters; boosting specifically does not.** Macro test error (%) on seven tabular datasets, mean ± propagated SEM over ten seeds. The LLM rule builder uses coverage tools and sees at most $M = 100$ examples per proposal; discrete search uses all training rows. Discrete set cover is a repaired variant selected on the development suite (Appendix C.2).

Harness	Model / rule search	$n = 100$	$n = 250$	$n = 600$	$n = 1{,}200$	$n = 2{,}000$
Boosting	Gemini 3.1 Flash-Lite	24.8 ± 0.3	23.2 ± 0.2	22.8 ± 0.1	$\mathbf{22.4 \pm 0.1}$	$\mathbf{22.2 \pm 0.1}$
Set cover	Gemini 3.1 Flash-Lite	$\mathbf{24.4 \pm 0.3}$	$\mathbf{23.0 \pm 0.1}$	$\mathbf{22.5 \pm 0.1}$	22.5 ± 0.1	22.4 ± 0.1
One-shot	Gemini 3.1 Flash-Lite	25.4 ± 0.4	23.9 ± 0.3	24.1 ± 0.3	24.8 ± 0.4	24.2 ± 0.4
Boosting	discrete search	$\mathbf{26.2 \pm 0.3}$	$\mathbf{23.9 \pm 0.2}$	$\mathbf{22.4 \pm 0.1}$	$\mathbf{22.0 \pm 0.1}$	$\mathbf{21.3 \pm 0.1}$
Set cover	discrete search	26.7 ± 0.4	24.6 ± 0.2	23.4 ± 0.1	22.8 ± 0.1	22.6 ± 0.1

versus 8.4% for SLIPPER; Appendix A.5). CLIPPER's edge over earlier rule learners on tabular data is also likely connected to its use of LLM background knowledge. To test this, we replace all the dataset, feature, class, and categorical-value names with meaningless identifiers in the tabular datasets. At $n = 250$, macro error on all eight datasets rises from 22.7 to 23.8%, a paired increase of 1.1 ± 0.2 points (Figure 4d).

4.3 From predicates to value-returning rules

A text rule can also return a value. We supply the calculator and requested entity key and ask the learner to extract its value from the note, for the ten most frequent entity keys (Appendix B). Each key has its own ten-round ensemble. Python and regex rules use $M = 20, 55, 110$, with ten seeds and no coverage tools. As a reference, we built a pool of regex rules by inspecting the training data with Anthropic's Claude Fable 5.1 as our coding agent, then selected rules from this fixed pool with the same scorer. We evaluate on 500 official-test calculator-row instances containing 1,120 values.

With a much weaker model, CLIPPER learns extractors that approach those written with a frontier coding agent. Python rules learned with Gemini 3.1 Flash-Lite at $M = 55$ reach 93.7 micro F1, within 1.6 points of the reference pool's 95.3 (Figure 4a), although Flash-Lite trails Fable 5.1 substantially on published benchmarks (Google DeepMind, 2026; Anthropic, 2026). The richer rule language helps: Python and regex rules are similarly precise (96.1 and 96.3%), but Python recovers more of the requested values (recall 91.4 versus 86.0%; Table 10).

4.4 The harness matters more than the model

One rule at a time. We first replace iterative rule construction with a one-shot prompt for a complete positive-class classifier, keeping the LLM model, sample cap, and tools fixed (Table 1). One-shot error is higher than boosted CLIPPER's at $n = 250$ (23.9 versus 23.2%), and the gap widens with more data: at $n = 1{,}200$, one-shot error rises to 24.8%, while boosting falls to 22.4%.

Another common way of learning rules iteratively is by "set covering": after each rule is learned, all examples it covers are removed, and a new rule is learned from the remaining data, approximating a greedy set-covering algorithm. Surprisingly, while replacing boosting with set covering negatively affects SLIPPER, the same harness change is sometimes slightly better when using the LLM-based rule builder.

Changing the model. Replacing the model inside CLIPPER changes error little (Table 3): all six models, including the small open-weight Qwen 3.5 9B, fall within 1.5 points. The model used for development, Gemini 3.1 Flash-Lite, nearly matches the best model, Sonnet 5, at a small fraction of its cost.

General-purpose coding agents.

Changing the harness and fixing the model leads to larger changes in performance. We also gave the whole learning task to general-purpose coding agents, which can use their built-in tools to inspect all 250 training examples, write and execute code, and so on (Table 2; Appendix E.1). Frontier models do well with their corresponding coding harnesses, but small models do not: Gemini 3.1 Flash-Lite and Haiku 4.5 are 3–4 points worse in their coding harnesses than inside CLIPPER (Tables 2

Table 2: **General-purpose coding agents.** Macro test error (%) on seven tabular datasets at $n = 250$, mean $\pm$ propagated SEM over three matched seeds. Runs in which an agent fit an existing learner were detected and corrected (Appendix E.2).

Harness	Model / rule search	Tools	Error (%)
Codex	**Sol**	default	$\mathbf{22.9 \pm 0.2}$
	Terra	default	23.5 ± 0.3
	Luna	default	24.2 ± 0.4
Claude Code	Opus 5	default	23.0 ± 0.4
	Sonnet 5	default	23.6 ± 0.3
	Haiku 4.5	default	28.1 ± 1.1
Gemini CLI	Gemini 3.5 Flash	default	24.0 ± 0.3
	Gemini 3.1 Flash-Lite	default	26.8 ± 0.9
Boosting	**Gemini 3.1 Flash-Lite**	coverage	$\mathbf{23.5 \pm 0.4}$
Boosting	discrete search	—	24.3 ± 0.4

Table 3: **Changing CLIPPER's model.** Boosting without coverage tools, $n = 250$, three matched seeds (Appendix E.5).

Model	Error (%)
Sonnet 5	$\mathbf{23.3 \pm 0.4}$
Gemini 3.1 Flash-Lite	23.4 ± 0.4
Gemini 2.5 Flash-Lite	23.9 ± 0.3
Haiku 4.5	24.2 ± 0.4
Qwen 3.5 9B	24.8 ± 0.5
Gemini 3.5 Flash	24.8 ± 0.6

and 3). Inside CLIPPER, Flash-Lite falls within the range of the frontier agents and beats full-data SLIPPER, at about \$0.04 per learning run, versus \$0.39–1.02 for Terra, Sonnet, Gemini 3.5, and Opus.

Qualitatively, the general-purpose coding agents behave quite differently than CLIPPER, as might be expected. They do not simply write rules in code: instead, they often write code for discrete search methods and use the results. In several cases agents tried to download and run existing interpretable learners (Appendix E).[8] We also note that some additional normally-strong coding models were tried but discarded because of "overthinking": i.e., taking many minutes and emitting thousands of thinking tokens on a single boosting round.

5 Discussion and conclusion

Our main interest in this work is understanding what LLMs can and cannot do. Although writing predictive rules is different from existing programming benchmarks, LLMs can be used to produce predictive rules, and combined with simple harnesses based on known rule-learning methods, can directly construct accurate interpretable classifiers. This result extends past work in LLM-driven construction of features to the harder problem of constructing complete classifiers, and using LLMs to write mechanistically interpretable executable classifiers provides a way of distilling LLM knowledge into these classifiers in a data-guided way.

An important outcome of this work is a new and better rule learner. This is of interest because interpretable models are a useful alternative to black-box models in some practically important settings—among them, auditing the behavior of LLMs!

Given the same data and same constraints, CLIPPER writes better rules than discrete search, on tabular data, where it is consistently better than or competitive with even full-data SLIPPER, and on text, where it wins by large margins. With the CLIPPER harness, an inexpensive model performs comparably to general-purpose agents built on frontier models, at a fraction of their cost. Using an LLM extends classical rule-learning algorithms beyond fixed attribute tests, and by doing so, we are able to construct text extractors with a small model that are highly competitive with extractors written interactively by a developer in collaboration with a frontier model.

Two natural next steps are fine-tuning small models to write better rules, with the score the harness already gives every proposed rule as a training signal, and scaling CLIPPER to larger samples of the training data. Both could make small, inexpensive models even stronger rule learners. Finally, learning interpretable classifiers is a useful benchmark for coding agents: it measures quality by held-out error, compares against mature specialist algorithms, and produces interpretable programs.

[8] We attempted to discourage this with our prompts, and believe that the results above are not from use of other baselines. While ways of combining the output of existing interpretable learners may be of interest, it is not the main focus of this work.

AI USE STATEMENT

We use an LLM to generate executable rules during CLIPPER training, as described in Section 3. We also used AI assistants to help find related work, plan and interpret experiments, construct the hand-written extraction pool, write and inspect parts of experiment code and outputs, and draft and edit manuscript sections. We have reviewed all AI-assisted work: code was tested, and every reported number, table, and figure is regenerated by script from saved per-run results and checked against them. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our experiments include health, credit, and recidivism data. Benchmark accuracy and executable rules do not establish suitability for decisions about individuals. MedCalc extraction uses supplied calculator identities and schemas, and silver reference values; it is not a validated clinical system.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A describe the learner, information access, and evaluation protocol. Appendices B and C report extension and development results, while Appendix D gives dataset-level tabular results. We release the codebase used for these experiments.

REFERENCES

Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. LLM-FE: Automated feature engineering for tabular data with LLMs as evolutionary optimizers. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=qvI35hkpOO.

Abhineet Agarwal, Yan Shuo Tan, Omer Ronen, Chandan Singh, and Bin Yu. Hierarchical shrinkage: Improving the accuracy and interpretability of tree-based methods. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 111–135, 2022.

Túlio C. Alberto, Johannes V. Lochter, and Tiago A. Almeida. TubeSpam: Comment spam filtering on YouTube. In *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 138–143, 2015.

Tiago A. Almeida, José María Gómez Hidalgo, and Akebo Yamakami. Contributions to the study of SMS spam filtering: New collection and results. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pp. 259–262, 2011.

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234): 1–78, 2018. URL https://jmlr.org/papers/v18/17-716.html.

Anthropic. Introducing Claude Fable 5.1 and Claude Mythos 5.1, 2026. URL https://www.anthropic.com/claude-fable-and-mythos-5-1. Accessed September 15, 2026.

Lucas Carrasco, Felipe Urrutia, and Andrés Abeliuk. Zero-shot decision tree construction via large language models, 2025. URL https://arxiv.org/abs/2501.16247.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pp. 38–45, 2020.

William W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123. Morgan Kaufmann, 1995.

William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 335–342. AAAI Press, 1999. URL https://cdn.aaai.org/AAAI/1999/AAAI99-049.pdf.

Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008. doi: 10.1214/07-AOAS148.

Daniela Agostina Gonzalez. Inflation research abstracts classification. UCI Machine Learning Repository, 2025. Dataset, donated February 16, 2025.

Google DeepMind. Gemini 3.5 Flash-Lite Model Card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-5-flash-lite/. Includes Gemini 3.1 Flash-Lite benchmark results. Accessed September 15, 2026.

Sungwon Han, Jinsung Yoon, Sercan Ö. Arik, and Tomas Pfister. Large language models can automatically engineer features for few-shot tabular learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17454–17479. PMLR, 2024. URL https://proceedings.mlr.press/v235/han24f.html.

Michael Heider, Helena Stegherr, Richard Nordsieck, and Jörg Hähner. Assessing model requirements for explainable ai: A template and exemplary case study. *Artificial Life*, 29(4):468–486, 11 2023. ISSN 1064-5462. doi: 10.1162/artl_a_00414. URL https://doi.org/10.1162/artl_a_00414.

Jingyun Jia, Chandan Singh, Rich Caruana, and Ben Lengerich. Selecting feature interactions for generalized additive models by distilling foundation models, 2026. URL https://arxiv.org/abs/2604.13332.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S. Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W. Safranek, Abid A. Anwar, Andrew Zhang, Aidan Gilson, Maxwell B. Singer, Amisha Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. MedCalc-Bench: Evaluating Large Language Models for Medical Calculations. In *Advances in Neural Information Processing Systems*, volume 37, pp. 84730–84745, 2024. doi: 10.52202/079017-2690. URL https://doi.org/10.52202/079017-2690.

Dimitrios Kotzias, Misha Denil, Nando de Freitas, and Padhraic Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 597–606, 2015.

Ádám Kovács, Nadia Verdha, and Gábor Recski. RuleChef: Grounding LLM task knowledge in human-editable rules, 2026. URL https://arxiv.org/abs/2607.01293.

Sven Kruschel, Nico Hambauer, Sven Weinzierl, Sandra Zilker, Mathias Kraus, and Patrick Zschech. Challenging the performance-interpretability trade-off: An evaluation of interpretable machine learning models. *Business & Information Systems Engineering*, 68(1):159–183, 2026. doi: 10.1007/s12599-024-00922-2.

Hariharan Manikandan, Yiding Jiang, and J. Zico Kolter. Language models are weak learners. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.14101.

METR. Brief independent investigation of agents' behavior, reasoning and collaboration in the openai / hugging face hacking incident. https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/, 08 2026.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16(none):1 – 85, 2022. doi: 10.1214/21-SS133. URL https://doi.org/10.1214/21-SS133.

Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999. doi: 10.1023/A:1007614523901.

Chandan Singh, Keyan Nasseri, Yan Shuo Tan, Tiffany Tang, and Bin Yu. imodels: A python package for fitting interpretable models. *Journal of Open Source Software*, 6(61):3192, 2021. doi: 10.21105/joss.03192.

Chandan Singh, Armin Askari, Rich Caruana, and Jianfeng Gao. Augmenting interpretable models with large language models during training. *Nature Communications*, 14(1):7913, 2023. doi: 10.1038/s41467-023-43713-1. URL https://www.nature.com/articles/s41467-023-43713-1.

Yan Shuo Tan, Chandan Singh, Keyan Nasseri, Abhineet Agarwal, James Duncan, Omer Ronen, Matthew Epland, Aaron Kornblith, and Bin Yu. Fast Interpretable Greedy-Tree Sums. *Proceedings of the National Academy of Sciences of the United States of America*, 122(7):e2310151122, 2025. doi: 10.1073/pnas.2310151122. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC11848335/.

Erica Zhang, Ryunosuke Goto, Naomi Sagan, Jurik Mutter, Nick Phillips, Ash Alizadeh, Kangwook Lee, Jose Blanchet, Mert Pilanci, and Robert Tibshirani. LLM-Lasso: A robust framework for domain-informed feature selection and regularization, 2025. URL https://arxiv.org/abs/2502.10648.

A Algorithms and evaluation details

A.1 Analysis of baselines

In Figure 2, SLIPPER is our re-implementation, which was also used for the LLM harness. The blue rule/tree baselines are GreedyRuleList, TaoTree, FPSkope, SkopeRules, and imodels' own SLIPPER reimplementation, all from imodels (Singh et al., 2021), plus RIPPER (via wittgenstein); ensembles are BoostedRules, RuleFit, FIGS, and HSTree from imodels. CORELS is a separate implementation. The datasets were adapted to be binary by making the positive class correspond to whatever category most directly corresponds to the more rare and/or interesting event, as follows: college, "Will a high-school student go to college?"; water potability, "Will the water be safe for consumption?"; stroke, "Will a patient suffer a stroke?', telco churn, "Will a customer leave the company?"; COMPAS, "Will a defendant recidivate?"; HELOC, " 'Will a client repay within 2 years?"; adult, "Will income exceed $50K/yr?"; bank marketing.csv, "Will a client subscribe?"

A.2 Classification algorithm

We use the following one-sided boosting procedure. Let $a(p, q) = \frac{1}{2}\log((p + \epsilon)/(q + \epsilon))$ and $G(p, q) = \sqrt{p} - \sqrt{q}$, with $\epsilon = 1/(2n)$. The positive class is the rarer training class under the default class ordering.

1. Initialize $D(i)$ from normalized example weights. Compute total positive and negative weights (P, Q), set $b = a(P, Q)$, and update $D(i) \propto D(i)\exp[-y_i b]$. Initialize the rule list to empty.
2. For each of the J rounds, draw up to M training examples without replacement, with selection probabilities proportional to D. The sampled wrappers retain relative weights, renormalized within the sample. $M = 0$ means full data. MedCalc classification additionally balances the first sample between classes where possible; subsequent samples use D directly.
3. Propose a predicate. Discrete SLIPPER grows and prunes a conjunction on this sample. CLIPPER receives its schema, labeled weighted rows, and the configured statistics or tools, then returns a Python predicate.
4. Evaluate the proposed predicate on all training examples under D to obtain (p, q). Recompute total class weights (P, Q). If $G(p, q) > \max\{G(P, Q), G(Q, P)\}$, retain the rule with confidence $a(p, q)$ and update covered weights by $\exp[-y_i a(p, q)]$. Otherwise update the default confidence by $a(P, Q)$ and reweight every example accordingly. Normalize D after either update.
5. Return the default score and retained rules. Sum the confidence of each matching rule and the default to predict. The implementation stores the default in smoothed coverage coordinates, with a numerical nonnegativity clamp when accumulating updates.

A failed model call, malformed function, or construction-time execution error can trigger up to three proposal attempts. If all fail, the classifier builder returns an always-false predicate and the usual default comparison applies. The tool-call limit is per attempt; it is not a bound on all calls across a fit. Tool-enabled attempts have a request limit of the tool budget plus five. The no-tools path has no iterative statistical-query interface. The coding model's temperature and provider seed are unset, so a learner seed does not fully determine its generated code.

A.3 Prompts and statistical tools

The classification system prompt is "You are a concise, careful Python programmer writing a single classification rule." The adopted task prompt asks for exactly one conjunction, favors a large cluster with more positives than negatives, and leaves other examples to subsequent rules. It requests explicit handling of missing values and standard-library code. Labels and weights are table columns, but are not keys available to the learned predicate. The displayed sample weights are rescaled to have mean one. The table format, full schema, quoted cell representation, and `task_variant_2` framing are common to the main CLIPPER conditions.

Table 4: Configurations. All six conditions rescore proposed rules on the full current training subset.

Configuration	Proposal sample
SLIPPER full	Full data
SLIPPER M100	$M = 100$
CLIPPER sample	$M = 100$
CLIPPER coverage	$M = 100$
CLIPPER hints	$M = 100$ + hints
CLIPPER hints+tools	$M = 100$ + hints

The coverage tool evaluates a single Python Boolean expression on each training row and returns the sums of current weights for covered positives and negatives. It supplies aggregate statistics, not extra labeled rows. The hints table supplies the best attribute test under discrete SLIPPER's criterion, including both sample and full-data coverage. The hints tool accepts a partial-rule filter and recomputes suggestions within that region. We provide the full templates and tool docstrings in the supplementary source files. Appendix G gives representative prompt excerpts, coverage-tool traces, and returned rules; full prompts accompany the source.

A.4 Tabular data and preprocessing

The suite builder uses a split seed of one, reserving a stratified 25% holdout for each task. Learning-curve seeds one through ten determine training subsets drawn from the remaining rows. Each method uses the same subset for a given dataset, size, and seed. The nine main sizes are 100, 175, 250, 400, 500, 600, 800, 1,200, and 2,000; a method may have only the five-size grid in its archived runs. College supports sizes through 600. The main figures fix the pool to the seven datasets that support 2,000.

Identifiers are removed from Stroke and Telco. Bank marketing excludes call `duration`, which is known after the call. Adult excludes `native-country`; COMPAS keeps nine demographic and criminal-history features and excludes the COMPAS score columns. The current builder applies this feature selection to the downloaded COMPAS rows; it does not implement an additional ProPublica cohort filter. Missing entries remain null in SLIPPER's input. Appendix tables use the prepared data, whose features can differ from the source survey's summary counts.

A.5 Text datasets

The text experiments (Figure 4b) use five two-class datasets: SMS spam (Almeida et al., 2011); Amazon, IMDb, and Yelp review sentences labeled positive or negative (Kotzias et al., 2015); and YouTube comment spam from five videos (Alberto et al., 2015). Spam makes up 13.4% of SMS messages and 51.4% of YouTube comments; the review sets are balanced. As for tabular data, each dataset keeps a stratified 25% test set, and every method trains on the same subset for each size (50 to 750) and seed (one to ten).

CLIPPER sees each document's raw text, may use regular expressions, and receives $M = 100$ examples per proposal without coverage tools. SLIPPER cannot read raw text, so it sees each document as its set of words and searches all training rows. HS-Tree (Agarwal et al., 2022) and the other `imodels` baselines use TF-IDF word weights over at most 800 words. HS-Tree has the lowest error of these baselines from $n = 200$; FIGS is lower at $n = 50$ and 100, by 0.8 and 0.2 points. Table 5 gives test error for each dataset.

A dataset published after the model's training cutoff. The five datasets above are long-public benchmarks that the model may have seen during training. We therefore also use Inflation Research Abstracts (Gonzalez, 2025), which was added to the UCI repository in February 2025, after the training cutoff of Gemini 3.1 Flash-Lite. It contains 1,138 abstracts of economics papers on inflation, labeled by whether the paper uses machine-learning or AI methods (20.9% positive). The abstracts themselves are older, but their labels were created for a 2025 study. We split the data and run every method as for the five datasets above, with 853 training and 285 test abstracts. CLIPPER reading

Table 5: **Text classification, by dataset.** Test error (%), mean ± SEM over ten seeds; bold marks the lowest error at each size. Macro rows average the five datasets.

Method	$n = 50$	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$	$n = 750$
SMS spam							
CLIPPER (no tools)	**8.4±1.5**	**4.9±1.0**	**3.5±0.5**	**2.9±0.4**	**2.2±0.1**	**2.6±0.2**	**2.0±0.3**
SLIPPER, full data	11.9±0.6	8.8±0.9	7.1±0.4	5.3±0.2	5.5±0.4	4.9±0.3	3.7±0.2
HS-Tree	12.5±0.5	10.5±0.6	8.2±0.3	7.3±0.2	7.3±0.4	6.8±0.3	6.1±0.3
Amazon							
CLIPPER (no tools)	**36.7±1.7**	**36.6±2.3**	**25.4±1.2**	**22.6±1.1**	**24.5±1.3**	**23.0±1.1**	**19.2±0.5**
SLIPPER, full data	43.0±1.6	43.8±1.4	35.4±1.2	30.9±0.9	31.0±1.0	32.6±1.1	29.0±0.5
HS-Tree	40.0±1.4	41.4±1.7	32.8±0.8	30.7±0.8	29.4±0.7	28.6±0.5	29.1±0.3
IMDb							
CLIPPER (no tools)	**40.7±1.2**	**37.0±1.6**	**33.7±2.0**	**30.1±0.7**	**28.1±0.7**	**27.4±0.7**	**24.8±0.5**
SLIPPER, full data	44.9±1.3	43.0±1.3	41.5±1.2	38.5±1.1	37.0±0.9	35.8±0.5	34.7±0.3
HS-Tree	45.0±1.0	42.8±0.9	41.2±0.9	40.3±0.7	38.6±0.8	39.1±0.9	37.2±0.0
Yelp							
CLIPPER (no tools)	**37.3±2.6**	**29.2±1.8**	**28.9±1.4**	**24.0±1.6**	**26.6±1.3**	**22.5±1.0**	**19.6±0.2**
SLIPPER, full data	44.6±1.5	40.8±1.0	34.4±1.7	34.1±1.1	31.2±1.1	30.2±0.6	28.8±0.3
HS-Tree	45.7±1.6	38.2±1.4	32.1±0.6	32.2±0.8	28.7±0.7	29.1±0.5	27.4±0.2
YouTube spam							
CLIPPER (no tools)	18.1±4.2	**8.7±0.7**	**8.4±0.5**	**9.1±0.8**	**8.4±0.7**	8.3±0.5	8.3±0.4
SLIPPER, full data	26.7±4.7	23.4±4.8	22.5±4.5	25.0±4.7	24.7±4.1	29.4±3.6	34.0±0.3
HS-Tree	**18.0±2.5**	12.3±0.5	10.1±0.5	9.7±0.3	9.0±0.5	**8.2±0.4**	**6.6±0.3**
Macro							
CLIPPER (no tools)	**28.2±1.1**	**23.3±0.7**	**20.0±0.6**	**17.7±0.4**	**18.0±0.4**	**16.8±0.3**	**14.8±0.2**
SLIPPER, full data	34.2±1.1	32.0±1.1	28.2±1.0	26.8±1.0	25.9±0.9	26.6±0.8	26.0±0.2
HS-Tree	32.2±0.7	29.1±0.5	24.9±0.3	24.1±0.3	22.6±0.3	22.4±0.2	21.3±0.1

Table 6: **Inflation Research Abstracts, published after the model's training cutoff.** Test error (%), mean ± SEM over ten seeds; bold marks the lowest error at each size. CLIPPER uses no coverage tools and $M = 100$. The `imodels` baselines use TF-IDF word weights, except CORELS, which uses word-presence indicators.

Method	$n = 50$	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$	$n = 750$
CLIPPER, raw text	**13.2±1.7**	**9.6±0.7**	**8.5±0.4**	**8.4±0.4**	**7.4±0.3**	**7.8±0.5**	**6.9±0.2**
CLIPPER, word sets	13.6±1.7	9.7±0.4	9.7±0.6	9.2±0.2	8.7±0.4	10.0±0.5	9.0±0.3
SLIPPER, full data	17.7±1.5	14.0±0.8	10.9±0.5	10.6±0.5	9.8±0.2	9.5±0.3	8.4±0.3
HS-Tree	23.1±1.9	18.1±1.3	15.8±1.2	16.2±0.9	14.4±0.5	13.4±0.8	13.4±0.4
Boosted stumps	20.2±1.2	16.7±1.1	12.5±0.5	11.8±0.4	11.5±0.4	11.6±0.2	11.3±0.3
CORELS	16.0±1.3	14.1±0.3	14.1±0.3	14.6±0.3	13.5±0.3	13.7±0.3	13.3±0.2
FIGS	21.8±2.1	19.1±1.8	16.0±1.4	13.6±0.9	13.0±0.4	12.1±0.7	12.3±0.4
TAO tree	20.0±2.2	19.2±1.9	16.4±1.3	14.2±0.7	12.6±0.4	13.3±0.7	12.2±0.4

the raw text has the lowest error at every training size, 6.9% at $n = 750$ versus 8.4% for full-data SLIPPER and 9.0% for CLIPPER reading word sets (Table 6).

A.6 Comparison with RuleChef on BANKING77

Recent systems also use an LLM to write rules. The closest to CLIPPER, RuleChef (Kovács et al., 2026), asks an LLM to write rules from labeled examples, but without boosting: it writes a small set of rules and then revises them to fix their mistakes. We compare the two on the benchmark RuleChef's authors use: five intents from BANKING77 (Casanueva et al., 2020). Both methods read the raw text and use the same model, Gemini 3.1 Flash-Lite, and RuleChef runs its authors' own benchmark script and settings.

Table 7: **BANKING77.** Test error (%), mean ± SEM over ten seeds; bold marks the lowest error at each size. When no RuleChef rule matches a query, it receives the most common intent in the training subset.

Method	$n = 50$	$n = 100$	$n = 200$	$n = 400$	$n = 600$
CLIPPER (tools)	**14.6±0.9**	**14.1±0.6**	**8.0±0.6**	**8.2±0.8**	**8.4±0.4**
CLIPPER (no tools)	16.8±1.1	16.0±1.1	11.0±0.5	10.4±0.6	9.1±0.6
RuleChef	53.7±3.0	36.1±3.2	39.7±3.9	37.7±2.5	34.9±1.7

CLIPPER's error is lower than RuleChef's in all 50 runs (five training sizes, ten seeds each), by 29.7 points on average with coverage tools and 27.8 points without them (Figure 4c and Table 7). From $n = 100$ on, RuleChef's error stays between 34.9 and 39.7%, while CLIPPER with tools reaches 8.0% at $n = 200$. RuleChef's rules label only about half of the test queries: after its first rounds of revision, it usually replaces its initial rules with one broad regular expression per intent.

The five intents are beneficiary not allowed, card arrival, disposable card limits, exchange rate, and pending cash withdrawal. Training queries come from BANKING77's official training split (685 queries), and the test set is all 200 official test queries for these intents, 40 each. For each size (50, 100, 200, 400, and 600) and seed (one to ten), both methods train on the same random subset.

CLIPPER reads each query's raw text, may use regular expressions, and receives $M = 100$ examples per proposal. We run it with the coverage tools used for MedCalc (Appendix B), at most six calls per proposal, and without them. As for MedCalc, it fits one two-class model per intent for six rounds and predicts the intent with the highest centered score.

We run RuleChef 0.2.2 through its authors' BANKING77 script, unchanged, with the settings of their published run: regular-expression rules, at most 100 rules, 200 examples shown to the model, up to 15 rounds of revision, and their agentic coordinator. The script writes rules from five examples per intent and revises them against the other $n - 25$ training queries. When no rule matches a test query, RuleChef returns no label; we assign the most common intent in that training subset, as SLIPPER's default rule does. Counting these queries as errors instead raises RuleChef's error to 47 to 66%.

A.7 Baseline settings and model size

We use `imodels` version 2.0.4 for the specialist baselines. Continuous inputs are median-imputed and categorical inputs one-hot encoded; preprocessing fits on the current training subset only. Binary-input learners such as CORELS additionally discretize continuous values into four quantile bins learned on that subset. Unknown test categories are ignored by the encoder. We use estimator defaults. RuleFit enables its own internal selection, and hierarchical shrinkage trees select shrinkage by internal cross-validation. We set a 600-second timeout per baseline fit.

Table 8: Baseline settings. These controls constrain different objects and do not define a common complexity budget.

Method	Settings
RuleFit	100 source trees, tree size 4, maximum rules 30, linear terms included, internal cross-validation enabled
FIGS	Maximum 12 splits
Boosted stumps	15 estimators, depth one
Greedy rule list	Maximum depth 5
SkopeRules	10 estimators, depth 3, minimum precision .5 and recall .01
CORELS	Regularization .01, maximum antecedent cardinality 2, minimum support .01, 10,000 search iterations

A curve seed is passed to an estimator only when its constructor exposes `random_state`; all seeds still identify shared training subsets. The separate `imodels` SLIPPER implementation failed in 50 of 380 dataset/size/seed cells, including every Water potability cell, and we exclude it from

the common aggregate. The failure is reported by its AdaBoost wrapper when the base classifier is worse than random.

At $n = 2{,}000$, the mean numbers of retained predicates on the seven datasets are 8.9 for full SLIPPER, 9.6 for CLIPPER sample, 11.8 for CLIPPER coverage, 16.0 for CLIPPER hints, and 17.4 for hints+tools. Each Python predicate can itself contain several operations, so these counts describe ensemble size rather than human effort.

A.8 Alternative harnesses

Covering removes covered examples and redistributes weight uniformly over survivors. Its positive-disjunction form accepts a rule only if $p > q$, and stops after one failed proposal unless a larger patience is specified. The discrete legacy condition combines FOIL-style growth with holdout pruning; its growth criterion differs from this acceptance gate.

One-shot is one outer round of this covering procedure. The LLM receives $M = 100$ examples and is asked for a complete disjunctive classifier inside one function. The cross-validated round budget becomes an approximate branch-count hint in its prompt. The entire function must pass the same $p > q$ gate, so a rejected proposal leaves a default-only classifier. The one-shot rule count indicates whether that function was retained; it does not count its internal branches. This changes hypothesis composition, acceptance, and total compute together with iteration.

A.9 Uncertainty and repetitions

Macro error averages the D dataset means equally, and its SEM is $\sqrt{\sum_d (\mathrm{SEM}_d/D)^2}$. SEM summarizes variation across the ten seeded training subsets and fits on a fixed holdout. It is not uncertainty over newly sampled datasets or new test sets. In particular, sizes from the same dataset are dependent and must not be treated as independent experiments in a pooled significance test. We compare observed means. For both MedCalc tasks, SEM measures variation over ten learner seeds with fixed training and evaluation rows. We compute calculator accuracy and extraction F1 separately for each seed, then average the ten scores. Resampling evaluation notes would estimate uncertainty conditional on each fitted model. The deterministic pool has no training-seed variation.

B MedCalc protocols and additional results

B.1 Data and calculator prediction

We use `nsk7153/MedCalc-Bench-Verified` at revision `591157b3343b4dda247294f9d929da4c75026fa8`, derived from MedCalc-Bench (Khandekar et al., 2024). Ten rows per calculator are held out from training before applying a training cap. The 55-way official test split has 1,100 rows, 20 per calculator. The realized training sizes at caps 1, 4, 16, and 64 are 55, 220, 870, and 3,302. Each method uses the same training rows at a given cap, with ten learner seeds.

SLIPPER and the other discrete learners cannot search over free text, so we represent each note by the set of words it contains. We lowercase the note, split it into maximal runs of the letters a–z, drop one-letter tokens, and remove duplicates. The vocabulary is built from the training rows alone: we keep words that appear in at least five training notes, up to the 5,000 most frequent, and remove all other words from training and test notes. A discrete rule condition then tests whether a given word is present, and a rule is a conjunction of such tests, such as "the note contains `wells` and `dvt`." This representation discards numbers, punctuation, word order, word counts, and therefore negation and phrases; a rule can test whether a note mentions `creatinine` but not its value. CLIPPER instead receives the raw note as the text attribute `NOTE` and can use its full content, including numbers and phrases. Neither method receives the question. The one-versus-rest runner uses six rounds and first-round stratified sampling, without internal round selection. A class prediction is the argmax of the 55 signed scores after removing their final learned default intercepts; lowest calculator ID breaks ties. Exact 55-way accuracy is primary. Calculator-family merging changes the prediction target. A sample cap limits rule proposal, while all retained rules receive full-training-set scoring and the default CLIPPER condition has coverage tools. We omit the optimization-criterion wording from the prompts for both calculator prediction and extraction.

Reading the raw notes, CLIPPER predicts the calculator more accurately than SLIPPER given the same sample of 20 examples per proposal at every training size from 220 notes on, and more accurately than full-data SLIPPER at 220 and 880 notes (Figure 5). Table 9 gives the exact accuracies, including $M = 100$.

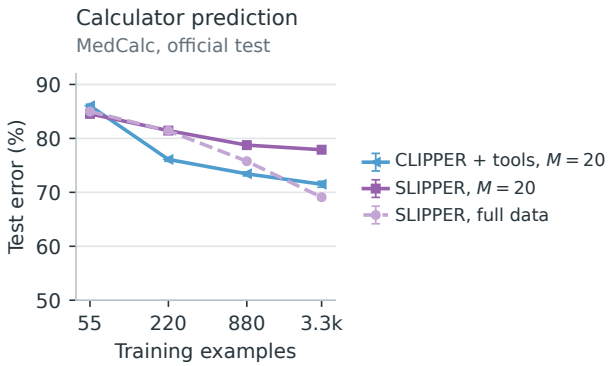


Figure 5: **Calculator prediction on MedCalc.** Test error (100 minus exact 55-way accuracy) on the official test split, mean and SEM over ten seeds. CLIPPER reads the raw notes; SLIPPER reads their word sets.

B.2 Value extraction

Extraction is conditioned on the true calculator and its requested fields. We keep the 33 calculators whose entity-key sets are consistent across their training and test rows, giving 34 distinct keys. Inconsistent omitted-versus-false annotations make the excluded schemas unsuitable for this exact-value task. The reference labels are silver `Relevant Entities` annotations. We select the ten most frequent keys after capping training at 150 rows per calculator, breaking ties alphabetically: sodium, weight, height, sex, bicarbonate, chloride, age, glucose, heart rate or pulse, and QT interval. These span 25 calculators. The official test split contains 500 instances and 1,120 requested values.

Table 9: Calculator prediction on the balanced official test. Exact accuracy (%), mean ± SEM over ten seeds per configuration and cap.

Cap	SLIPPER			CLIPPER (raw notes)	
	Full	$M = 20$	$M = 100$	$M = 20$	$M = 100$
1	15.0 ± 0.3	15.5 ± 0.5	15.0 ± 0.3	13.9 ± 0.3	10.1 ± 0.5
4	18.6 ± 0.3	18.6 ± 0.3	20.1 ± 0.4	23.9 ± 0.3	22.1 ± 0.3
16	24.2 ± 0.3	21.2 ± 0.3	25.8 ± 0.5	26.6 ± 0.4	25.9 ± 0.3
64	30.9 ± 0.4	22.1 ± 0.4	25.9 ± 0.4	28.5 ± 0.6	28.2 ± 0.4

We use ten learner seeds on fixed training rows. Appendix B.3 gives the training and prediction procedure.

Canonicalization folds number formatting, case, spacing, and unit spelling. It does not perform unit conversion or general synonym matching. Precision, recall, and micro F1 compare predicted and gold sets of key/value pairs. An incorrect returned value contributes both a false positive and a missed gold value; abstention contributes only a missed gold value. Complete-row accuracy requires exact equality of all selected values for that row.

We wrote the hand-written regex pools using Anthropic's Claude Fable 5.1 as our coding agent. We inspected the training examples, checked the patterns' precision and coverage on them, and selected rules from the resulting pool using the same scoring procedure. All ten selected keys have a pool. For all three approaches, we train each model once using only the training data, then evaluate it on the official test split.

Table 10: Extraction official-test results (%), mean ± SEM over ten seeds. The deterministic pool is evaluated once. "All selected" is complete-row accuracy.

Rule source	Precision	Recall	F1	All selected
Hand-written pool	97.0	93.7	95.3	91.0
Python, $M = 20$	95.6 ± 0.3	90.4 ± 0.6	92.9 ± 0.3	84.7 ± 1.3
Python, $M = 55$	96.1 ± 0.2	91.4 ± 0.5	93.7 ± 0.2	87.0 ± 0.8
Python, $M = 110$	96.1 ± 0.2	90.6 ± 0.5	93.3 ± 0.3	84.7 ± 1.1
Regex, $M = 20$	96.3 ± 0.2	86.4 ± 1.1	91.1 ± 0.6	77.7 ± 2.0
Regex, $M = 55$	96.3 ± 0.1	86.0 ± 0.6	90.8 ± 0.3	77.4 ± 1.2
Regex, $M = 110$	96.2 ± 0.2	85.8 ± 1.2	90.7 ± 0.7	77.2 ± 2.1

On official test, the pool has 1,049 true positives, 33 false positives, and 71 false negatives, with all selected fields correct on 455 of 500 rows.

B.3 Extraction with SLIPPER and CLIPPER

Each training example contains `NOTE`, the supplied `CALC_ID`, and the requested `ENTITY`; its target is that entity's annotated value. We pool examples of the same entity across calculators and learn one ensemble per entity key. SLIPPER supplies the boosting procedure: sampling examples, scoring rules, assigning confidences, and updating weights. The hand-written baseline selects rules from a fixed regex pool; CLIPPER generates a new candidate with the coding model.

For a key with n training examples, initialize $D_1(i) = 1/n$ and an empty ensemble. The default prediction is abstention. Use a budget of ten rounds:

1. **Propose a rule.** CLIPPER draws up to $M \in \{20, 55, 110\}$ examples without replacement using D_t. The prompt contains their inputs, gold values, weights, and schema. Gemini 3.1 Flash-Lite returns either a Python function `extract(x)` or a regex pattern, flags, and output template. Both return a value or `None`. Python can perform arithmetic; the regex template combines captured text and literals. Neither arm uses coverage tools. The pool baseline instead considers every hand-written rule.

2. **Score on all training examples.** For a candidate r_t, let $s_t(i)$ be $+1$ for a correct returned value, -1 for an incorrect return, and 0 for abstention, comparing values after canonicalization. Compute

$$p_t = \sum_{i:s_t(i)=1} D_t(i), \qquad q_t = \sum_{i:s_t(i)=-1} D_t(i). \tag{3}$$

   The pool baseline chooses the candidate with the largest $1 + \sqrt{p_t} - \sqrt{q_t}$. CLIPPER's candidate is scored by the same procedure, including on training examples outside its prompt sample.

3. **Retain and reweight.** If $p_t > q_t$, retain the rule with confidence α_t and update

$$\alpha_t = \frac{1}{2} \log \frac{p_t + \epsilon}{q_t + \epsilon}, \qquad D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t s_t(i)}}{Z_t}, \qquad \epsilon = \frac{1}{2n}, \tag{4}$$

   where Z_t normalizes the weights. Before normalization, correct returns multiply an example's weight by $e^{-\alpha_t}$, incorrect returns by e^{α_t}, and abstentions leave it unchanged. If $p_t \leq q_t$, CLIPPER discards the candidate and continues with unchanged weights. The pool baseline stops, since its best candidate failed the same test.

A failed CLIPPER proposal has up to two retries; exhausting them adds no rule that round. Execution errors and one-second timeouts count as abstentions. Duplicate retained rules are merged by adding their confidences.

At prediction time, run the saved rules for each requested key. Group their outputs by canonical value, sum the confidences within each group, and return the value with the largest sum. If no rule returns a value, abstain. This step uses no LLM calls. We apply the fitted ensembles to the official test split, then join the key predictions back into each calculator row for scoring.

B.4 Example extraction prompts and rules

These excerpts show CLIPPER's extraction prompts and three generated rules. We show only the relevant text and fields from each sampled example.

Captures and literals. The regex prompt specifies how the learner will execute the returned rule:

```
Task: write a rule as a REGEX SPEC -- structured fields, not code.  The learner
runs it as
    m = re.search(pattern, x['NOTE'], flags)
and builds the value from `template`, so fill in these three fields:
```

Here `i` ignores case and `1` inserts the first captured group. For age, a sampled example and the returned rule are:

```
ENTITY: age
NOTE: "An 89-year-old Japanese woman..."
GOLD: [89, "years"]

pattern: (\d+(?:\.\d+)?)-year-old
flags: i
template: [1, "years"]
```

The capture supplies the number; the template supplies the unit. For sex, another prompt and returned rule illustrate a literal output:

```
ENTITY: sex
NOTE: "A 31-year-old woman..."
GOLD: "Female"

pattern: \b(?:woman|female|girl)\b
flags: i
template: ["Female"]
```

This rule returns `Female` for any of the three matched words and abstains when none occurs.

Python and arithmetic. The Python prompt asks for a value-returning function:

```
Task: write a rule in the form of a Python function
    def extract(x: dict): ...
that returns the VALUE asked for by `x['ENTITY']`, read out of the note text in
`x['NOTE']`, for a meaningful cluster of examples -- and returns None on every
other example.
```

A platelet-count rule from the earlier 34-key experiment includes the following branch, with `re` already imported:

```
note = x.get('NOTE')
if not note or x.get('ENTITY') != 'Platelet count':
    return None
m = re.search(
    r'platelet(?: count)?(?: of)?\s+(\d+(?:\.\d+)?)\s*k[/\s]?(?:μ|u)l',
    note, re.IGNORECASE)
if m:
    return [float(m.group(1)) * 1000, 'μL']
```

For the training-note excerpt below, the branch returns the annotated value:

```
NOTE: "...platelet count of 233 k/μL..."
GOLD / RETURNED VALUE: [233000.0, "μL"]
```

Python expresses the multiplication by 1,000; a capture template can only combine matched text and literals.

C Development experiments and covering controls

C.1 Development suite and configuration selection

Our development suite contains Abalone, Breast cancer Wisconsin, Cylinder bands, German credit, Hypothyroid, Move1, Obesity, Pima diabetes, Portuguese student performance, and Titanic. It was selected for binary targets, meaningful attribute names, and varied domains. Nine datasets use a fixed, stratified 25% holdout with split seed one; Move1 retains its native split. Prompt and configuration comparisons use ten training seeds. The holdouts in this suite guide development and are distinct from the tabular evaluation suite in the main text.

Abalone labels age through a threshold of ten rings. Obesity collapses the three obesity categories and removes height and weight. Student performance uses final grade below ten and excludes the two preceding grades. German credit codes are decoded to meaningful attribute values. Missing data remain explicit; the Pima source's zero-valued measurements were not converted to null. The Obesity source includes synthetic augmentation. These choices are part of the task definitions we evaluate.

Table 11: Development configurations in the full-data comparison. Macro test error (%) averages ten dataset means, with ten seeds per dataset. Rows are different configurations, not successive additions to one system.

Configuration	Macro error
Discrete SLIPPER, full data	18.1
CLIPPER, table sample, no tools (earlier prompt)	20.0
Per-attribute SLIPPER condition tools	19.2
One-conjunction framing with coverage tool	19.0
Revised coverage-tool prompt	18.8
Condition-hints table with iterative queries	**17.7**

Sample size. In the dedicated sample-size comparison, increasing M from 100 to 200 and 400 gives macro errors of 20.5, 20.8, and 20.9%. These values come from a different configuration comparison than the table above. The larger prompt samples do not improve average error. Coverage-retrieval probes additionally record counting errors on long samples, motivating the coverage tools. The sample-size curve alone does not identify the cause of the degradation.

Prompt representation and task framing. Replacing per-example dictionaries with a table reduced prompt tokens by about 71%, with a 0.5-point macro-error increase. We retained the full schema, quoted cell values, and table format in our main configuration. Most wording changes had small measured effects. The one-conjunction-per-round task framing improved macro error by 1.0 points relative to its prompt control. Its instruction is still a prompt preference, rather than a syntactic check on generated predicates.

Statistical feedback. Per-attribute optimizers expose the best full-data threshold or category. The condition table packages these suggestions with sample and full-data coverage; its tool form recomputes them within a partial rule's coverage. The latter achieved the lowest development macro in the table, 17.7%. The separate initial-table comparison gave 18.8% and varied across datasets. We therefore evaluate initial hints and iterative queries as separate feedback conditions in the main experiments.

C.2 Repairs to discrete covering

The legacy covering learner uses a positive disjunction and accepts rules only when their residual positive coverage exceeds negative coverage. Two repairs change how discrete search reaches that gate. Gate-aware sizing retains only pruned generalizations that still pass the gate and prefers the more specific rule on ties. Linear growth replaces the FOIL-style criterion with $p - q + 1$, matching the change in covered training error. Two further variants learn ordered decision lists with per-rule consequents; these change the hypothesis class as well as the search.

The six conditions each contain 440 dataset/size/seed cells: ten development datasets at sizes 100, 200, 300, and 400, plus the four datasets that support size 800, with ten seeds in every group. The pooled cell mean below weights these 440 cells equally. It is not the seven-dataset test macro used in the main tables.

Table 12: Completed ten-seed discrete covering comparisons on the development suite. Cells with no retained rule predict the fallback class.

Condition	Pooled error (%)	Zero-rule cells / 440
Legacy covering	23.8	50
Gate-aware sizing	24.1	11
Linear growth	23.3	31
Gate-aware sizing + linear growth	**23.0**	1
Decision list, FOIL growth	23.9	20
Decision list, linear growth	23.5	71

The combined repair improves on legacy covering in 206 cells, worsens in 186, and ties in 48. It reduces empty fits substantially, but at size 800 its pooled error is 22.5%, compared with 21.2% for legacy covering. Thus the repair's effect depends on training size. On the seven main test datasets, ten seeds give $24.6 \pm 0.2\%$ error at $n = 250$ and $22.8 \pm 0.1\%$ at $n = 1{,}200$ (Table 13), using the same training subsets and training-only cross-validated round budgets.

C.3 Covering patience with the LLM proposer

Patience three skips a failed proposal and permits a new proposal under the same residual distribution; patience one stops immediately. Both conditions have 380 cells on the tabular evaluation suite, covering the five-size grid 100, 250, 600, 1,200, and 2,000 with ten seeds per feasible dataset and size. Their pooled cell errors are 23.0% and 22.9% for patience one and three. Zero-rule fits fall from 76 to 56, while total model-call cost rises from \$22.93 to \$44.11. These pooled summaries have a changing dataset pool; Table 13 recomputes the size-specific comparisons on the fixed seven-dataset pool. The results show more retained rules with mixed accuracy effects, and qualify a blanket claim that boosting outperforms covering for an LLM rule proposer.

Table 13: All outer-loop conditions on the seven main tabular datasets. Error (%), mean $\pm$ propagated SEM over ten seeds. Discrete search uses full training data; the LLM proposer uses $M = 100$. Patience 3 permits up to two further proposals after a rejected rule. The last row uses gate-aware pruning and linear growth.

Harness / rule proposer	$n = 250$	$n = 1{,}200$
Boosting / SLIPPER	23.9 ± 0.2	$\mathbf{22.0 \pm 0.1}$
Legacy covering / SLIPPER	25.7 ± 0.3	23.9 ± 0.2
Boosting / CLIPPER coverage	23.2 ± 0.2	22.4 ± 0.1
Covering / CLIPPER, patience 1	$\mathbf{23.0 \pm 0.1}$	22.5 ± 0.1
Covering / CLIPPER, patience 3	23.5 ± 0.2	22.2 ± 0.1
One-shot / CLIPPER coverage	23.9 ± 0.3	24.8 ± 0.4
One-shot / CLIPPER hints	26.6 ± 0.4	26.8 ± 0.3
Repaired covering / SLIPPER	24.6 ± 0.2	22.8 ± 0.1

D DATASET-LEVEL TABULAR RESULTS

We plot the fixed seven-dataset macro used in the main figures and tables for every CLIPPER variant (Figure 6), followed by dataset-level results that include College where available.

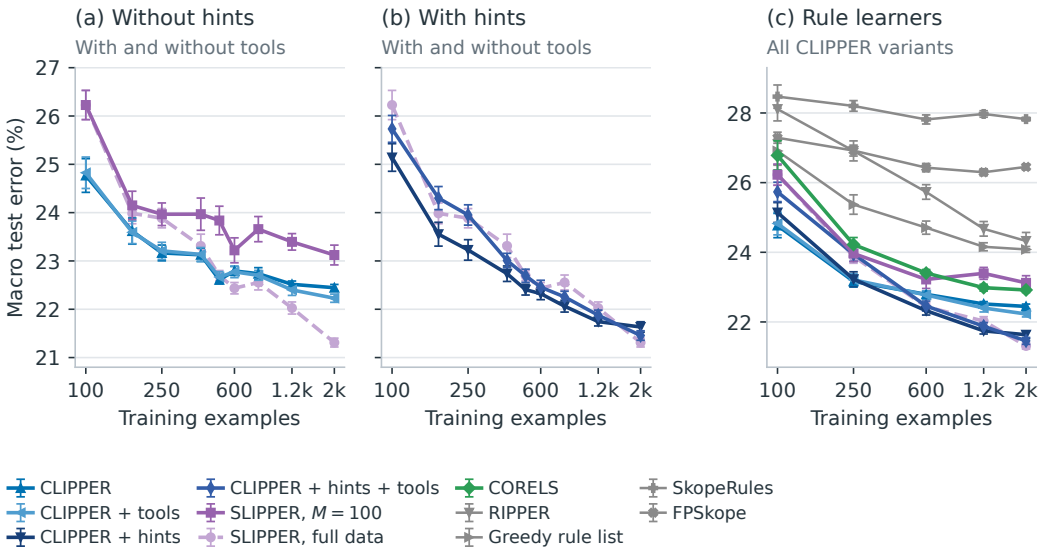


Figure 6: **All CLIPPER variants on tabular data.** The curves of Figure 3 for every CLIPPER variant: (a) without hints and (b) with hints, each with and without coverage tools, and (c) all four variants against established rule learners. Macro test error on seven tabular datasets (mean and propagated SEM, ten seeds).

Each cell is test-error percentage, mean ± SEM over seeds. Macro error is the unweighted mean across datasets; its SEM is $\sqrt{\sum_d (\mathrm{SEM}_d/D)^2}$. In the dataset-level tables, bold values mark the lowest mean in each row among the methods shown in that panel, including ties. Panels showing only one method are not bolded. Bold does not indicate statistical significance. We compute mean rank across all methods in each comparison, including methods shown in separate panels.

D.1 DISCRETE BASELINES

Table 14: Discrete baselines at $n = 250$, panel 1 of 4.

Dataset / statistic	RuleFit	Boosted stumps	SLIPPER
Adult	18.1 ± 0.4	$\mathbf{17.1 \pm 0.4}$	19.3 ± 0.7
Bank marketing	12.1 ± 0.1	$\mathbf{11.4 \pm 0.2}$	12.0 ± 0.3
College	$\mathbf{15.6 \pm 0.6}$	16.1 ± 0.6	17.2 ± 1.2
COMPAS	33.7 ± 0.2	$\mathbf{33.1 \pm 0.4}$	34.3 ± 0.5
HELOC	31.0 ± 0.4	$\mathbf{30.6 \pm 0.3}$	32.7 ± 0.7
Stroke	6.4 ± 0.3	6.4 ± 0.4	$\mathbf{5.9 \pm 0.4}$
Telco churn	$\mathbf{21.7 \pm 0.4}$	22.6 ± 0.4	24.1 ± 0.6
Water potability	39.0 ± 0.5	42.2 ± 0.7	$\mathbf{38.9 \pm 0.3}$
Macro	$\mathbf{22.2 \pm 0.1}$	22.4 ± 0.2	23.1 ± 0.2
Mean rank	3.1	**2.9**	4.1

Table 15: Discrete baselines at $n = 250$, panel 2 of 4.

Dataset / statistic	CORELS	FIGS	TAO tree
Adult	$\mathbf{19.5 \pm 0.7}$	20.3 ± 0.5	20.2 ± 0.6
Bank marketing	$\mathbf{11.3 \pm 0.2}$	13.5 ± 0.3	13.7 ± 0.3
College	24.9 ± 0.4	$\mathbf{16.4 \pm 0.5}$	19.1 ± 0.7
COMPAS	36.0 ± 0.7	$\mathbf{35.6 \pm 0.6}$	36.0 ± 0.6
HELOC	$\mathbf{33.1 \pm 0.5}$	34.3 ± 0.6	34.5 ± 0.5
Stroke	$\mathbf{4.9 \pm 0.0}$	8.3 ± 0.3	8.9 ± 0.3
Telco churn	$\mathbf{22.8 \pm 0.5}$	24.0 ± 0.5	23.7 ± 0.6
Water potability	$\mathbf{42.0 \pm 0.6}$	42.4 ± 0.9	44.1 ± 1.3
Macro	$\mathbf{24.3 \pm 0.2}$	24.4 ± 0.2	25.0 ± 0.2
Mean rank	**4.4**	6.3	6.9

Table 16: Discrete baselines at $n = 250$, panel 3 of 4.

Dataset / statistic	Greedy rule list	HS tree	FPSkope
Adult	25.4 ± 1.0	20.6 ± 0.5	$\mathbf{20.5 \pm 0.6}$
Bank marketing	$\mathbf{11.5 \pm 0.2}$	14.4 ± 0.3	11.7 ± 0.0
College	23.4 ± 0.8	$\mathbf{19.0 \pm 0.5}$	50.0 ± 0.0
COMPAS	$\mathbf{36.2 \pm 0.8}$	37.6 ± 0.5	41.3 ± 0.6
HELOC	$\mathbf{31.5 \pm 0.5}$	35.6 ± 0.6	47.6 ± 0.1
Stroke	5.6 ± 0.6	9.3 ± 0.4	$\mathbf{4.9 \pm 0.0}$
Telco churn	26.9 ± 0.6	23.8 ± 0.6	$\mathbf{22.6 \pm 0.2}$
Water potability	40.6 ± 1.0	44.7 ± 1.0	$\mathbf{40.0 \pm 0.6}$
Macro	$\mathbf{25.1 \pm 0.3}$	25.6 ± 0.2	29.8 ± 0.1
Mean rank	6.0	8.1	**5.9**

Table 17: Discrete baselines at $n = 250$, panel 4 of 4.

Dataset / statistic	SkopeRules
Adult	22.8 ± 0.6
Bank marketing	11.7 ± 0.2
College	50.0 ± 0.0
COMPAS	44.9 ± 0.1
HELOC	47.8 ± 0.0
Stroke	6.1 ± 0.5
Telco churn	25.1 ± 0.7
Water potability	39.0 ± 0.1
Macro	30.9 ± 0.1
Mean rank	7.3

Table 18: Discrete baselines at $n = 1200$, panel 1 of 4.

Dataset / statistic	Boosted stumps	RuleFit	FIGS
Adult	15.8 ± 0.1	$\mathbf{15.6 \pm 0.2}$	16.1 ± 0.2
Bank marketing	$\mathbf{10.7 \pm 0.0}$	11.3 ± 0.1	11.3 ± 0.1
COMPAS	32.9 ± 0.4	32.8 ± 0.3	$\mathbf{32.4 \pm 0.3}$

Table 18: Discrete baselines at $n = 1200$ (continued).

Dataset / statistic	Boosted stumps	RuleFit	FIGS
HELOC	28.9 ± 0.3	$\mathbf{28.4 \pm 0.1}$	29.5 ± 0.3
Stroke	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$	5.8 ± 0.2
Telco churn	$\mathbf{20.9 \pm 0.2}$	21.5 ± 0.1	21.5 ± 0.3
Water potability	37.9 ± 0.3	38.1 ± 0.2	$\mathbf{37.4 \pm 0.6}$
Macro	$\mathbf{21.7 \pm 0.1}$	21.8 ± 0.1	22.0 ± 0.1
Mean rank	**2.6**	3.4	3.7

Table 19: Discrete baselines at $n = 1200$, panel 2 of 4.

Dataset / statistic	SLIPPER	TAO tree	HS tree
Adult	$\mathbf{15.6 \pm 0.1}$	16.1 ± 0.2	16.4 ± 0.2
Bank marketing	$\mathbf{11.1 \pm 0.1}$	11.4 ± 0.1	11.7 ± 0.2
COMPAS	$\mathbf{32.9 \pm 0.4}$	33.0 ± 0.4	33.3 ± 0.4
HELOC	$\mathbf{30.5 \pm 0.7}$	30.6 ± 0.4	31.4 ± 0.4
Stroke	$\mathbf{4.9 \pm 0.0}$	5.8 ± 0.2	6.3 ± 0.3
Telco churn	$\mathbf{21.8 \pm 0.2}$	22.0 ± 0.3	21.9 ± 0.4
Water potability	$\mathbf{37.5 \pm 0.3}$	37.7 ± 0.6	38.6 ± 0.4
Macro	$\mathbf{22.0 \pm 0.1}$	22.3 ± 0.1	22.8 ± 0.1
Mean rank	**3.1**	5.9	7.4

Table 20: Discrete baselines at $n = 1200$, panel 3 of 4.

Dataset / statistic	CORELS	Greedy rule list	FPSkope
Adult	$\mathbf{17.8 \pm 0.0}$	22.5 ± 0.5	19.2 ± 0.3
Bank marketing	11.1 ± 0.1	$\mathbf{11.0 \pm 0.1}$	11.7 ± 0.0
COMPAS	$\mathbf{34.8 \pm 0.3}$	36.1 ± 0.5	39.8 ± 0.6
HELOC	31.6 ± 0.3	$\mathbf{30.3 \pm 0.2}$	47.8 ± 0.0
Stroke	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$
Telco churn	22.5 ± 0.5	26.5 ± 0.0	$\mathbf{21.7 \pm 0.1}$
Water potability	38.3 ± 0.1	$\mathbf{37.8 \pm 0.4}$	39.0 ± 0.0
Macro	$\mathbf{23.0 \pm 0.1}$	24.2 ± 0.1	26.3 ± 0.1
Mean rank	6.4	**5.8**	7.8

Table 21: Discrete baselines at $n = 1200$, panel 4 of 4.

Dataset / statistic	SkopeRules
Adult	23.8 ± 0.2
Bank marketing	11.6 ± 0.1
COMPAS	45.1 ± 0.0
HELOC	47.8 ± 0.0
Stroke	5.4 ± 0.2
Telco churn	23.2 ± 0.5
Water potability	38.9 ± 0.1
Macro	28.0 ± 0.1

Table 21: Discrete baselines at $n = 1200$ (continued).

Dataset / statistic	SkopeRules
Mean rank	8.9

D.2 CLIPPER AND SLIPPER

Table 22: CLIPPER and SLIPPER at $n = 250$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER no tools	CLIPPER tools
Adult	$\mathbf{17.4 \pm 0.5}$	17.8 ± 0.9	17.6 ± 0.7
Bank marketing	11.3 ± 0.2	$\mathbf{11.0 \pm 0.1}$	11.3 ± 0.2
College	$\mathbf{17.2 \pm 0.6}$	18.8 ± 0.9	19.0 ± 0.4
COMPAS	$\mathbf{32.5 \pm 0.4}$	33.5 ± 0.3	33.5 ± 0.3
HELOC	31.5 ± 0.3	$\mathbf{31.2 \pm 0.3}$	33.1 ± 0.4
Stroke	6.2 ± 0.5	$\mathbf{5.4 \pm 0.3}$	6.1 ± 0.4
Telco churn	22.4 ± 0.6	23.4 ± 0.2	$\mathbf{22.2 \pm 0.4}$
Water potability	41.3 ± 1.0	39.9 ± 0.4	$\mathbf{38.8 \pm 0.7}$
Macro	$\mathbf{22.5 \pm 0.2}$	22.6 ± 0.2	22.7 ± 0.2
Mean rank	2.9	**2.5**	3.0

Table 23: CLIPPER and SLIPPER at $n = 250$, panel 2 of 2.

Dataset / statistic	SLIPPER $M = 100$	CLIPPER suggestions + tools	SLIPPER
Adult	20.2 ± 1.0	$\mathbf{17.7 \pm 0.5}$	19.3 ± 0.7
Bank marketing	$\mathbf{11.6 \pm 0.0}$	11.8 ± 0.3	12.0 ± 0.3
College	$\mathbf{16.3 \pm 0.5}$	16.6 ± 0.6	17.2 ± 1.2
COMPAS	34.6 ± 0.6	$\mathbf{34.1 \pm 0.4}$	34.3 ± 0.5
HELOC	$\mathbf{32.4 \pm 0.9}$	33.5 ± 0.3	32.7 ± 0.7
Stroke	$\mathbf{5.6 \pm 0.4}$	6.5 ± 0.6	5.9 ± 0.4
Telco churn	$\mathbf{23.4 \pm 0.4}$	23.7 ± 0.9	24.1 ± 0.6
Water potability	39.9 ± 0.4	40.4 ± 0.6	$\mathbf{38.9 \pm 0.3}$
Macro	$\mathbf{23.0 \pm 0.2}$	$\mathbf{23.0 \pm 0.2}$	23.1 ± 0.2
Mean rank	**3.8**	4.5	4.4

Table 24: CLIPPER and SLIPPER at $n = 1200$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER suggestions + tools	SLIPPER
Adult	$\mathbf{15.4 \pm 0.1}$	$\mathbf{15.4 \pm 0.1}$	15.6 ± 0.1
Bank marketing	$\mathbf{10.9 \pm 0.1}$	11.0 ± 0.1	11.1 ± 0.1
COMPAS	$\mathbf{31.8 \pm 0.2}$	32.0 ± 0.2	32.9 ± 0.4
HELOC	$\mathbf{29.6 \pm 0.3}$	30.3 ± 0.4	30.5 ± 0.7
Stroke	5.1 ± 0.2	5.3 ± 0.2	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{21.2 \pm 0.3}$	21.4 ± 0.3	21.8 ± 0.2
Water potability	38.1 ± 0.3	37.8 ± 0.4	$\mathbf{37.5 \pm 0.3}$

Table 24: CLIPPER and SLIPPER at $n = 1200$ (continued).

Dataset / statistic	CLIPPER suggestions	CLIPPER suggestions + tools	SLIPPER
Macro	$\mathbf{21.7 \pm 0.1}$	21.9 ± 0.1	22.0 ± 0.1
Mean rank	**2.0**	2.9	3.1

Table 25: CLIPPER and SLIPPER at $n = 1200$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER no tools	SLIPPER $M = 100$
Adult	$\mathbf{16.0 \pm 0.2}$	16.1 ± 0.1	17.9 ± 0.7
Bank marketing	11.0 ± 0.1	$\mathbf{10.8 \pm 0.0}$	11.7 ± 0.0
COMPAS	$\mathbf{32.8 \pm 0.2}$	$\mathbf{32.8 \pm 0.1}$	33.2 ± 0.4
HELOC	32.4 ± 0.5	$\mathbf{30.6 \pm 0.3}$	33.8 ± 0.6
Stroke	5.1 ± 0.2	5.1 ± 0.2	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{22.1 \pm 0.4}$	23.5 ± 0.1	23.6 ± 0.6
Water potability	$\mathbf{37.4 \pm 0.4}$	38.8 ± 0.2	38.7 ± 0.2
Macro	$\mathbf{22.4 \pm 0.1}$	22.5 ± 0.1	23.4 ± 0.2
Mean rank	**3.9**	**3.9**	5.3

D.3 Alternative harnesses

Table 26: Alternative harnesses at $n = 250$, panel 1 of 3.

Dataset / statistic	CLIPPER tools	CLIPPER set cover	CLIPPER suggestions + tools
Adult	$\mathbf{17.6 \pm 0.7}$	17.7 ± 0.6	17.7 ± 0.5
Bank marketing	$\mathbf{11.3 \pm 0.2}$	$\mathbf{11.3 \pm 0.3}$	11.8 ± 0.3
College	19.0 ± 0.4	22.3 ± 0.8	$\mathbf{16.6 \pm 0.6}$
COMPAS	33.5 ± 0.3	$\mathbf{33.3 \pm 0.2}$	34.1 ± 0.4
HELOC	33.1 ± 0.4	$\mathbf{31.2 \pm 0.4}$	33.5 ± 0.3
Stroke	6.1 ± 0.4	$\mathbf{5.4 \pm 0.4}$	6.5 ± 0.6
Telco churn	$\mathbf{22.2 \pm 0.4}$	22.9 ± 0.3	23.7 ± 0.9
Water potability	$\mathbf{38.8 \pm 0.7}$	39.5 ± 0.3	40.4 ± 0.6
Macro	$\mathbf{22.7 \pm 0.2}$	22.9 ± 0.2	23.0 ± 0.2
Mean rank	**2.9**	3.0	4.8

Table 27: Alternative harnesses at $n = 250$, panel 2 of 3.

Dataset / statistic	SLIPPER	One-shot CLIPPER	Discrete set cover
Adult	19.3 ± 0.7	$\mathbf{18.2 \pm 0.9}$	25.4 ± 0.6
Bank marketing	12.0 ± 0.3	11.7 ± 0.2	$\mathbf{11.4 \pm 0.1}$
College	$\mathbf{17.2 \pm 1.2}$	28.4 ± 1.2	20.3 ± 0.8
COMPAS	$\mathbf{34.3 \pm 0.5}$	35.3 ± 1.3	36.7 ± 1.1
HELOC	32.7 ± 0.7	$\mathbf{32.3 \pm 0.4}$	33.3 ± 0.9
Stroke	5.9 ± 0.4	5.3 ± 0.4	$\mathbf{4.9 \pm 0.0}$

Table 27: Alternative harnesses at $n = 250$ (continued).

Dataset / statistic	SLIPPER	One-shot CLIPPER	Discrete set cover
Telco churn	$\mathbf{24.1 \pm 0.6}$	24.9 ± 0.5	25.5 ± 1.0
Water potability	$\mathbf{38.9 \pm 0.3}$	39.9 ± 0.4	42.5 ± 1.1
Macro	$\mathbf{23.1 \pm 0.2}$	24.5 ± 0.3	25.0 ± 0.3
Mean rank	**4.4**	4.8	5.8

Table 28: Alternative harnesses at $n = 250$, panel 3 of 3.

Dataset / statistic	Suggestions set cover	One-shot suggestions
Adult	$\mathbf{20.5 \pm 1.2}$	23.4 ± 0.7
Bank marketing	$\mathbf{11.1 \pm 0.2}$	11.7 ± 0.0
College	$\mathbf{36.9 \pm 2.5}$	40.0 ± 3.4
COMPAS	$\mathbf{36.4 \pm 1.1}$	41.7 ± 1.4
HELOC	$\mathbf{30.9 \pm 0.3}$	38.1 ± 2.3
Stroke	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{25.2 \pm 0.4}$	27.0 ± 0.6
Water potability	39.6 ± 0.4	$\mathbf{39.3 \pm 0.3}$
Macro	$\mathbf{25.7 \pm 0.4}$	28.2 ± 0.6
Mean rank	**4.3**	6.3

Table 29: Alternative harnesses at $n = 1200$, panel 1 of 3.

Dataset / statistic	CLIPPER suggestions + tools	SLIPPER	CLIPPER tools
Adult	$\mathbf{15.4 \pm 0.1}$	15.6 ± 0.1	16.0 ± 0.2
Bank marketing	$\mathbf{11.0 \pm 0.1}$	11.1 ± 0.1	$\mathbf{11.0 \pm 0.1}$
COMPAS	$\mathbf{32.0 \pm 0.2}$	32.9 ± 0.4	32.8 ± 0.2
HELOC	$\mathbf{30.3 \pm 0.4}$	30.5 ± 0.7	32.4 ± 0.5
Stroke	5.3 ± 0.2	$\mathbf{4.9 \pm 0.0}$	5.1 ± 0.2
Telco churn	$\mathbf{21.4 \pm 0.3}$	21.8 ± 0.2	22.1 ± 0.4
Water potability	37.8 ± 0.4	37.5 ± 0.3	$\mathbf{37.4 \pm 0.4}$
Macro	$\mathbf{21.9 \pm 0.1}$	22.0 ± 0.1	22.4 ± 0.1
Mean rank	**2.3**	2.4	3.4

Table 30: Alternative harnesses at $n = 1200$, panel 2 of 3.

Dataset / statistic	CLIPPER set cover	Discrete set cover	Suggestions set cover
Adult	$\mathbf{16.3 \pm 0.2}$	21.4 ± 0.6	21.1 ± 1.0
Bank marketing	$\mathbf{11.0 \pm 0.1}$	11.1 ± 0.1	11.4 ± 0.1
COMPAS	$\mathbf{33.2 \pm 0.4}$	34.4 ± 0.6	36.5 ± 1.2
HELOC	$\mathbf{30.5 \pm 0.2}$	30.8 ± 0.4	$\mathbf{30.5 \pm 0.2}$
Stroke	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{22.3 \pm 0.2}$	26.0 ± 0.5	24.7 ± 0.3
Water potability	39.1 ± 0.2	39.0 ± 0.6	$\mathbf{38.9 \pm 0.1}$

Table 30: Alternative harnesses at $n = 1200$ (continued).

Dataset / statistic	CLIPPER set cover	Discrete set cover	Suggestions set cover
Macro	$\mathbf{22.5 \pm 0.1}$	23.9 ± 0.2	24.0 ± 0.2
Mean rank	**4.4**	5.4	4.9

Table 31: Alternative harnesses at $n = 1200$, panel 3 of 3.

Dataset / statistic	One-shot CLIPPER	One-shot suggestions
Adult	$\mathbf{18.8 \pm 1.0}$	24.1 ± 0.0
Bank marketing	$\mathbf{11.3 \pm 0.1}$	11.7 ± 0.0
COMPAS	$\mathbf{36.5 \pm 1.2}$	43.9 ± 0.8
HELOC	37.7 ± 2.4	$\mathbf{37.5 \pm 2.2}$
Stroke	$\mathbf{4.9 \pm 0.0}$	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{25.6 \pm 0.5}$	26.7 ± 0.2
Water potability	$\mathbf{39.0 \pm 0.0}$	$\mathbf{39.0 \pm 0.0}$
Macro	$\mathbf{24.8 \pm 0.4}$	26.8 ± 0.3
Mean rank	**6.1**	7.1

D.4 Complete summary learning-curve results

Table 32: Complete summary learning-curve results at $n = 100$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER tools	CLIPPER no tools
Adult	19.0 ± 0.5	$\mathbf{18.1 \pm 0.5}$	18.3 ± 0.7
Bank marketing	12.1 ± 0.3	11.8 ± 0.3	$\mathbf{11.5 \pm 0.2}$
College	$\mathbf{20.4 \pm 1.3}$	22.9 ± 1.0	25.7 ± 1.4
COMPAS	$\mathbf{36.4 \pm 1.0}$	37.6 ± 1.5	37.6 ± 1.1
HELOC	36.9 ± 1.0	36.4 ± 1.3	$\mathbf{35.9 \pm 1.9}$
Stroke	5.2 ± 0.2	5.0 ± 0.1	$\mathbf{4.9 \pm 0.0}$
Telco churn	25.2 ± 1.1	$\mathbf{23.6 \pm 0.5}$	24.7 ± 0.5
Water potability	41.2 ± 0.7	41.3 ± 0.7	$\mathbf{40.5 \pm 0.6}$
Macro	$\mathbf{24.5 \pm 0.3}$	24.6 ± 0.3	24.9 ± 0.4
Mean rank	3.0	2.4	**2.1**

Table 33: Complete summary learning-curve results at $n = 100$, panel 2 of 2.

Dataset / statistic	CLIPPER suggestions + tools	SLIPPER
Adult	$\mathbf{19.6 \pm 0.4}$	20.9 ± 0.7
Bank marketing	$\mathbf{13.3 \pm 0.6}$	13.8 ± 0.6
College	$\mathbf{20.2 \pm 1.0}$	20.3 ± 1.1
COMPAS	$\mathbf{37.7 \pm 1.0}$	39.1 ± 0.9
HELOC	$\mathbf{37.1 \pm 1.0}$	39.0 ± 1.3
Stroke	6.5 ± 0.4	$\mathbf{5.3 \pm 0.2}$

Table 33: Complete summary learning-curve results at $n = 100$ (continued).

Dataset / statistic	CLIPPER suggestions + tools	SLIPPER
Telco churn	$\mathbf{23.9 \pm 0.5}$	24.5 ± 0.7
Water potability	42.0 ± 0.9	$\mathbf{41.0 \pm 0.8}$
Macro	$\mathbf{25.0 \pm 0.3}$	25.5 ± 0.3
Mean rank	**3.6**	3.9

Table 34: Complete summary learning-curve results at $n = 175$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	SLIPPER	CLIPPER no tools
Adult	18.2 ± 0.7	18.9 ± 0.4	$\mathbf{18.0 \pm 0.8}$
Bank marketing	12.0 ± 0.3	12.5 ± 0.5	$\mathbf{11.2 \pm 0.2}$
College	$\mathbf{18.6 \pm 1.0}$	18.8 ± 1.1	22.0 ± 1.7
COMPAS	$\mathbf{32.8 \pm 0.5}$	34.8 ± 0.9	33.6 ± 0.4
HELOC	$\mathbf{32.7 \pm 0.7}$	33.6 ± 0.8	32.8 ± 1.3
Stroke	5.2 ± 0.3	$\mathbf{5.1 \pm 0.2}$	$\mathbf{5.1 \pm 0.2}$
Telco churn	$\mathbf{23.5 \pm 0.7}$	23.9 ± 0.5	23.7 ± 0.3
Water potability	40.5 ± 1.0	$\mathbf{39.3 \pm 0.3}$	41.0 ± 0.9
Macro	$\mathbf{22.9 \pm 0.2}$	23.3 ± 0.2	23.4 ± 0.3
Mean rank	**2.1**	3.4	2.9

Table 35: Complete summary learning-curve results at $n = 175$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER suggestions + tools
Adult	$\mathbf{16.8 \pm 0.4}$	18.5 ± 0.7
Bank marketing	$\mathbf{11.6 \pm 0.3}$	12.5 ± 0.4
College	23.0 ± 1.6	$\mathbf{18.1 \pm 0.9}$
COMPAS	$\mathbf{33.4 \pm 0.3}$	34.1 ± 0.6
HELOC	34.4 ± 1.3	$\mathbf{34.2 \pm 0.7}$
Stroke	$\mathbf{5.4 \pm 0.3}$	6.2 ± 0.4
Telco churn	$\mathbf{22.7 \pm 0.4}$	23.5 ± 0.4
Water potability	$\mathbf{40.9 \pm 0.6}$	$\mathbf{40.9 \pm 1.0}$
Macro	$\mathbf{23.5 \pm 0.3}$	$\mathbf{23.5 \pm 0.2}$
Mean rank	**2.9**	3.8

Table 36: Complete summary learning-curve results at $n = 250$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER no tools	CLIPPER tools
Adult	$\mathbf{17.4 \pm 0.5}$	17.8 ± 0.9	17.6 ± 0.7
Bank marketing	11.3 ± 0.2	$\mathbf{11.0 \pm 0.1}$	11.3 ± 0.2

Table 36: Complete summary learning-curve results at $n = 250$ (continued).

Dataset / statistic	CLIPPER suggestions	CLIPPER no tools	CLIPPER tools
College	$\mathbf{17.2 \pm 0.6}$	18.8 ± 0.9	19.0 ± 0.4
COMPAS	$\mathbf{32.5 \pm 0.4}$	33.5 ± 0.3	33.5 ± 0.3
HELOC	31.5 ± 0.3	$\mathbf{31.2 \pm 0.3}$	33.1 ± 0.4
Stroke	6.2 ± 0.5	$\mathbf{5.4 \pm 0.3}$	6.1 ± 0.4
Telco churn	22.4 ± 0.6	23.4 ± 0.2	$\mathbf{22.2 \pm 0.4}$
Water potability	41.3 ± 1.0	39.9 ± 0.4	$\mathbf{38.8 \pm 0.7}$
Macro	$\mathbf{22.5 \pm 0.2}$	22.6 ± 0.2	22.7 ± 0.2
Mean rank	2.5	**2.4**	2.6

Table 37: Complete summary learning-curve results at $n = 250$, panel 2 of 2.

Dataset / statistic	CLIPPER suggestions + tools	SLIPPER
Adult	$\mathbf{17.7 \pm 0.5}$	19.3 ± 0.7
Bank marketing	$\mathbf{11.8 \pm 0.3}$	12.0 ± 0.3
College	$\mathbf{16.6 \pm 0.6}$	17.2 ± 1.2
COMPAS	$\mathbf{34.1 \pm 0.4}$	34.3 ± 0.5
HELOC	33.5 ± 0.3	$\mathbf{32.7 \pm 0.7}$
Stroke	6.5 ± 0.6	$\mathbf{5.9 \pm 0.4}$
Telco churn	$\mathbf{23.7 \pm 0.9}$	24.1 ± 0.6
Water potability	40.4 ± 0.6	$\mathbf{38.9 \pm 0.3}$
Macro	$\mathbf{23.0 \pm 0.2}$	23.1 ± 0.2
Mean rank	**3.8**	**3.8**

Table 38: Complete summary learning-curve results at $n = 400$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions + tools	CLIPPER suggestions	SLIPPER
Adult	16.4 ± 0.2	$\mathbf{15.9 \pm 0.1}$	17.0 ± 0.4
Bank marketing	11.8 ± 0.2	$\mathbf{11.4 \pm 0.2}$	11.9 ± 0.3
College	$\mathbf{14.3 \pm 0.7}$	16.3 ± 0.4	15.2 ± 0.7
COMPAS	34.1 ± 0.4	$\mathbf{33.1 \pm 0.4}$	33.6 ± 0.6
HELOC	31.5 ± 0.6	$\mathbf{30.9 \pm 0.4}$	33.3 ± 1.4
Stroke	5.6 ± 0.3	5.5 ± 0.3	$\mathbf{5.1 \pm 0.2}$
Telco churn	$\mathbf{22.1 \pm 0.3}$	22.2 ± 0.4	23.2 ± 0.4
Water potability	39.6 ± 0.5	40.2 ± 0.8	$\mathbf{39.1 \pm 0.5}$
Macro	$\mathbf{21.9 \pm 0.2}$	$\mathbf{21.9 \pm 0.1}$	22.3 ± 0.2
Mean rank	2.9	**2.4**	3.1

Table 39: Complete summary learning-curve results at $n = 400$, panel 2 of 2.

Dataset / statistic	CLIPPER no tools	CLIPPER tools
Adult	$\mathbf{16.0 \pm 0.2}$	$\mathbf{16.0 \pm 0.2}$
Bank marketing	11.4 ± 0.2	$\mathbf{11.3 \pm 0.1}$
College	$\mathbf{17.9 \pm 0.5}$	19.5 ± 1.0
COMPAS	34.5 ± 0.5	$\mathbf{34.1 \pm 0.4}$
HELOC	$\mathbf{30.9 \pm 0.3}$	32.9 ± 0.6
Stroke	5.6 ± 0.3	$\mathbf{5.4 \pm 0.3}$
Telco churn	23.2 ± 0.1	$\mathbf{22.2 \pm 0.3}$
Water potability	40.3 ± 0.7	$\mathbf{39.9 \pm 0.5}$
Macro	$\mathbf{22.5 \pm 0.1}$	22.7 ± 0.2
Mean rank	3.6	**3.0**

Table 40: Complete summary learning-curve results at $n = 500$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions + tools	CLIPPER suggestions	SLIPPER
Adult	16.7 ± 0.3	$\mathbf{16.2 \pm 0.4}$	16.5 ± 0.2
Bank marketing	11.4 ± 0.1	$\mathbf{11.1 \pm 0.1}$	11.5 ± 0.2
College	$\mathbf{13.6 \pm 0.6}$	16.7 ± 0.6	15.2 ± 0.8
COMPAS	32.9 ± 0.3	$\mathbf{32.4 \pm 0.3}$	33.2 ± 0.4
HELOC	31.3 ± 0.5	$\mathbf{30.8 \pm 0.4}$	31.7 ± 0.4
Stroke	5.3 ± 0.3	5.0 ± 0.1	$\mathbf{4.9 \pm 0.0}$
Telco churn	21.7 ± 0.2	$\mathbf{21.4 \pm 0.2}$	22.2 ± 0.3
Water potability	39.6 ± 0.6	39.9 ± 0.3	$\mathbf{38.8 \pm 0.2}$
Macro	$\mathbf{21.6 \pm 0.1}$	21.7 ± 0.1	21.8 ± 0.1
Mean rank	3.3	**2.3**	3.2

Table 41: Complete summary learning-curve results at $n = 500$, panel 2 of 2.

Dataset / statistic	CLIPPER no tools	CLIPPER tools
Adult	16.3 ± 0.3	$\mathbf{15.8 \pm 0.2}$
Bank marketing	11.4 ± 0.2	$\mathbf{11.2 \pm 0.2}$
College	$\mathbf{18.7 \pm 0.8}$	19.8 ± 0.5
COMPAS	$\mathbf{32.8 \pm 0.2}$	33.5 ± 0.4
HELOC	$\mathbf{30.4 \pm 0.2}$	32.1 ± 0.3
Stroke	$\mathbf{4.9 \pm 0.0}$	5.3 ± 0.2
Telco churn	22.9 ± 0.2	$\mathbf{21.8 \pm 0.2}$
Water potability	39.6 ± 0.3	$\mathbf{38.9 \pm 0.5}$
Macro	$\mathbf{22.1 \pm 0.1}$	22.3 ± 0.1
Mean rank	**2.8**	3.5

Table 42: Complete summary learning-curve results at $n = 600$, panel 1 of 2.

Dataset / statistic	SLIPPER	CLIPPER suggestions + tools	CLIPPER suggestions
Adult	16.6 ± 0.1	$\mathbf{16.0 \pm 0.1}$	$\mathbf{16.0 \pm 0.2}$
Bank marketing	11.1 ± 0.2	$\mathbf{11.0 \pm 0.1}$	$\mathbf{11.0 \pm 0.1}$
College	13.6 ± 0.5	$\mathbf{13.5 \pm 0.5}$	15.7 ± 0.6
COMPAS	32.9 ± 0.4	$\mathbf{32.3 \pm 0.4}$	32.7 ± 0.5
HELOC	31.2 ± 0.5	31.1 ± 0.3	$\mathbf{30.6 \pm 0.3}$
Stroke	$\mathbf{4.9 \pm 0.0}$	5.4 ± 0.3	5.0 ± 0.1
Telco churn	21.9 ± 0.2	21.7 ± 0.3	$\mathbf{21.4 \pm 0.2}$
Water potability	$\mathbf{38.5 \pm 0.5}$	39.8 ± 0.7	39.7 ± 0.6
Macro	$\mathbf{21.3 \pm 0.1}$	$\mathbf{21.3 \pm 0.1}$	21.5 ± 0.1
Mean rank	2.9	2.6	**2.4**

Table 43: Complete summary learning-curve results at $n = 600$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER no tools
Adult	$\mathbf{16.0 \pm 0.3}$	16.6 ± 0.4
Bank marketing	11.0 ± 0.1	$\mathbf{10.9 \pm 0.1}$
College	$\mathbf{17.3 \pm 0.5}$	18.0 ± 0.7
COMPAS	$\mathbf{33.6 \pm 0.3}$	$\mathbf{33.6 \pm 0.3}$
HELOC	32.3 ± 0.4	$\mathbf{30.8 \pm 0.4}$
Stroke	5.1 ± 0.2	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{22.0 \pm 0.2}$	23.4 ± 0.1
Water potability	$\mathbf{39.4 \pm 0.5}$	$\mathbf{39.4 \pm 0.2}$
Macro	$\mathbf{22.1 \pm 0.1}$	22.2 ± 0.1
Mean rank	3.6	**3.4**

Table 44: Complete summary learning-curve results at $n = 800$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER suggestions + tools	SLIPPER
Adult	$\mathbf{15.7 \pm 0.3}$	16.0 ± 0.2	16.6 ± 0.2
Bank marketing	$\mathbf{11.0 \pm 0.1}$	11.2 ± 0.1	11.3 ± 0.2
COMPAS	$\mathbf{32.6 \pm 0.4}$	32.9 ± 0.3	33.2 ± 0.5
HELOC	$\mathbf{30.4 \pm 0.4}$	31.3 ± 0.5	31.2 ± 0.6
Stroke	$\mathbf{4.9 \pm 0.1}$	5.0 ± 0.1	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{20.7 \pm 0.2}$	20.9 ± 0.1	21.8 ± 0.4
Water potability	39.1 ± 0.5	$\mathbf{38.5 \pm 0.5}$	38.8 ± 0.6
Macro	$\mathbf{22.1 \pm 0.1}$	22.3 ± 0.1	22.6 ± 0.2
Mean rank	**1.9**	2.9	3.6

Table 45: Complete summary learning-curve results at $n = 800$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER no tools
Adult	$\mathbf{16.0 \pm 0.2}$	16.1 ± 0.2
Bank marketing	11.0 ± 0.1	$\mathbf{10.9 \pm 0.1}$
COMPAS	$\mathbf{33.1 \pm 0.3}$	33.8 ± 0.5
HELOC	33.1 ± 0.5	$\mathbf{30.8 \pm 0.6}$
Stroke	5.3 ± 0.2	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{21.7 \pm 0.2}$	23.1 ± 0.2
Water potability	$\mathbf{38.8 \pm 0.3}$	39.6 ± 0.4
Macro	$\mathbf{22.7 \pm 0.1}$	$\mathbf{22.7 \pm 0.1}$
Mean rank	**3.3**	3.4

Table 46: Complete summary learning-curve results at $n = 1200$, panel 1 of 2.

Dataset / statistic	CLIPPER suggestions	CLIPPER suggestions + tools	SLIPPER
Adult	$\mathbf{15.4 \pm 0.1}$	$\mathbf{15.4 \pm 0.1}$	15.6 ± 0.1
Bank marketing	$\mathbf{10.9 \pm 0.1}$	11.0 ± 0.1	11.1 ± 0.1
COMPAS	$\mathbf{31.8 \pm 0.2}$	32.0 ± 0.2	32.9 ± 0.4
HELOC	$\mathbf{29.6 \pm 0.3}$	30.3 ± 0.4	30.5 ± 0.7
Stroke	5.1 ± 0.2	5.3 ± 0.2	$\mathbf{4.9 \pm 0.0}$
Telco churn	$\mathbf{21.2 \pm 0.3}$	21.4 ± 0.3	21.8 ± 0.2
Water potability	38.1 ± 0.3	37.8 ± 0.4	$\mathbf{37.5 \pm 0.3}$
Macro	$\mathbf{21.7 \pm 0.1}$	21.9 ± 0.1	22.0 ± 0.1
Mean rank	**1.9**	2.7	3.1

Table 47: Complete summary learning-curve results at $n = 1200$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER no tools
Adult	$\mathbf{16.0 \pm 0.2}$	16.1 ± 0.1
Bank marketing	11.0 ± 0.1	$\mathbf{10.8 \pm 0.0}$
COMPAS	$\mathbf{32.8 \pm 0.2}$	$\mathbf{32.8 \pm 0.1}$
HELOC	32.4 ± 0.5	$\mathbf{30.6 \pm 0.3}$
Stroke	$\mathbf{5.1 \pm 0.2}$	$\mathbf{5.1 \pm 0.2}$
Telco churn	$\mathbf{22.1 \pm 0.4}$	23.5 ± 0.1
Water potability	$\mathbf{37.4 \pm 0.4}$	38.8 ± 0.2
Macro	$\mathbf{22.4 \pm 0.1}$	22.5 ± 0.1
Mean rank	3.7	**3.6**

Table 48: Complete summary learning-curve results at $n = 2000$, panel 1 of 2.

Dataset / statistic	SLIPPER	CLIPPER suggestions + tools	CLIPPER suggestions
Adult	15.3 ± 0.1	15.3 ± 0.1	$\mathbf{14.9 \pm 0.1}$
Bank marketing	$\mathbf{10.8 \pm 0.0}$	10.9 ± 0.0	$\mathbf{10.8 \pm 0.0}$
COMPAS	31.8 ± 0.2	31.6 ± 0.3	$\mathbf{31.1 \pm 0.3}$
HELOC	$\mathbf{29.1 \pm 0.2}$	29.7 ± 0.4	30.1 ± 0.5
Stroke	$\mathbf{4.9 \pm 0.0}$	5.0 ± 0.2	5.0 ± 0.2
Telco churn	$\mathbf{21.1 \pm 0.3}$	21.8 ± 0.2	22.0 ± 0.4
Water potability	36.2 ± 0.5	$\mathbf{35.9 \pm 0.4}$	37.5 ± 0.2
Macro	$\mathbf{21.3 \pm 0.1}$	21.5 ± 0.1	21.6 ± 0.1
Mean rank	**1.9**	2.3	2.6

Table 49: Complete summary learning-curve results at $n = 2000$, panel 2 of 2.

Dataset / statistic	CLIPPER tools	CLIPPER no tools
Adult	$\mathbf{15.6 \pm 0.2}$	$\mathbf{15.6 \pm 0.1}$
Bank marketing	$\mathbf{10.9 \pm 0.0}$	$\mathbf{10.9 \pm 0.1}$
COMPAS	$\mathbf{32.4 \pm 0.2}$	32.8 ± 0.1
HELOC	32.4 ± 0.3	$\mathbf{30.7 \pm 0.4}$
Stroke	$\mathbf{4.9 \pm 0.1}$	$\mathbf{4.9 \pm 0.1}$
Telco churn	$\mathbf{21.9 \pm 0.2}$	23.3 ± 0.1
Water potability	$\mathbf{37.6 \pm 0.3}$	38.9 ± 0.1
Macro	$\mathbf{22.2 \pm 0.1}$	22.4 ± 0.1
Mean rank	**3.9**	4.4

E ADDITIONAL EXPERIMENT DETAILS AND AGENT BEHAVIOR

The experiments in Tables 3 and 2 and Figure 4d use $n = 250$ and the tabular protocol of Section 4. The main results use seven datasets, except anonymization, which uses all eight; the tables below also include College. Anonymization uses ten seeds, and model and agent comparisons use seeds 1–3, including for reference learners. Compared methods receive the same training subsets and fixed test holdouts. Macro error weights dataset means equally, with SEM $\sqrt{\sum_d \mathrm{SEM}_d^2}/D$ for D datasets. For differences, we compute each dataset's SEM from paired seed differences and propagate it in the same way. The bars describe variation across fits on these fixed tasks and holdouts.

E.1 RULE CONSTRUCTION IN GENERAL-PURPOSE CODING HARNESSES

Each agent receives `train.csv`, `schema.json`, and a task requesting a readable Python classifier, `classify(x: dict) -> str`. It can inspect all 250 training examples, run exploratory scripts, and evaluate candidate classifiers. The objective is ordinary classification error, with equal cost for either class. Each run uses a fresh Docker container managed by Sandy, with a hashed workspace name, disabled web-search tools, and restricted network access that still permits Python package downloads. Test data remain outside the container. CLIPPER instead requests one predicate at a time from at most $M = 100$ examples, with full-training scoring and, in the coverage configuration, tools for querying coverage.

Claude Code 2.1.272 used high effort, a $2 session cap, and 40 turns. Codex CLI 0.154.0 used high effort and a 1,800-second cap; one original Sol run (Stroke, seed 3) was retried with 3,000 seconds. Gemini CLI 0.60.0 used 40 turns for Flash-Lite and 80 for Gemini 3.5, after 40 turns proved insufficient on Stroke. A requested Gemini 3.8 configuration was reported by the CLI as Gemini 3.5 and is labeled accordingly. These comparisons evaluate complete systems, including their prompts, data access, and computation budgets. Failed sessions were rerun to complete the three-seed grid; the table reports the saved classifiers.

Table 50: Complete coding-agent comparison at $n = 250$. Dataset columns show mean test error (%) over three matched seeds; macro columns include propagated SEM. The seven-dataset pool excludes College.

System	College	Water	Stroke	Telco	COMPAS	HELOC	Adult	Bank	Macro 8	Macro 7
Sol	17.9	41.4	**5.3**	22.0	33.2	31.6	**15.8**	11.3	$\mathbf{22.3 \pm 0.3}$	$\mathbf{22.9 \pm 0.2}$
Opus 5	20.7	**38.2**	5.9	22.5	34.5	31.2	17.4	11.2	22.7 ± 0.4	23.0 ± 0.4
Terra	23.6	39.0	5.8	23.8	34.7	31.8	18.5	**10.9**	23.5 ± 0.4	23.5 ± 0.3
CLIPPER (gemini 3.1 flash-lite)	18.8	38.4	6.9	22.2	33.3	33.3	19.0	11.5	22.9 ± 0.4	23.5 ± 0.4
Sonnet 5	22.9	41.7	6.7	22.3	33.4	**30.9**	18.3	12.0	23.5 ± 0.3	23.6 ± 0.3
Gemini 3.5 CLI	17.9	43.7	6.2	**21.8**	33.7	31.7	16.8	14.1	23.2 ± 0.3	24.0 ± 0.3
Luna	18.7	43.6	**5.3**	24.4	**32.2**	32.0	19.5	12.6	23.5 ± 0.3	24.2 ± 0.4
SLIPPER	**16.3**	38.8	6.9	23.9	33.2	33.1	21.4	12.9	23.3 ± 0.4	24.3 ± 0.4
Gemini 3.1 CLI	32.4	43.9	15.4	27.7	34.4	32.5	22.2	11.7	27.5 ± 0.9	26.8 ± 0.9
Haiku 4.5	25.9	44.4	22.2	24.4	**32.2**	33.2	22.7	17.4	27.8 ± 1.0	28.1 ± 1.1

E.2 SPECIFICATION GAMING THROUGH UNINTENDED LEARNER FITTING

Our intended task was for an agent to construct rules by examining examples and evaluating candidates. The original prompt expressed this intention through restrictions on the submitted classifier: a short sequence of if/then rules, with no imports, training-example lookup tables, or learned weight vectors. These restrictions did not explicitly prohibit fitting another learner in a helper script. An agent could therefore satisfy the output requirements while handing rule induction to an existing algorithm.

The container logs record whether each run reached the Python package index. Under the original prompt, Sol did so in 10 of its 24 runs, installing scikit-learn and, in one College run, `wittgenstein`, a Python implementation of RIPPER. On HELOC seed 2, for example, it fitted a cross-validated decision tree and transcribed its splits as if/then rules; the submitted classifier contained ordinary comparisons and no library import, and achieved 26.6% test error. The strategy generally worked for Sol. Compared with the best other system on the same dataset and seed, its error was 1.4 points lower on average in these 10 runs and 1.0 point higher in the remaining 14. Other

agents attempted the same strategy with little success. Opus 5, Sonnet 5, and Gemini 3.5 Flash installed scikit-learn in 3, 3, and 2 runs, and in those runs trailed the best other system by about 6, 3, and 3 points. Luna and Terra attempted to install scikit-learn but never reached the package index; Gemini 3.1 Flash-Lite and Haiku 4.5 never reached it either. This is specification gaming at its core: the program had the requested form, but the way it was constructed bypassed the intended experiment.

E.3 Making the intended restriction explicit

We repeated Sol's 24 runs with an instruction forbidding both invoking an existing learning algorithm and implementing one in a helper script. Descriptive statistics and tests of agent-proposed rules remained available. Helper scripts also had to carry a comment explaining their purpose.

On the seven main-text datasets, error rose from 20.7 ± 0.3 to $22.9 \pm 0.2\%$, a paired increase of 2.2 ± 0.4 percentage points. Sol's mean advantage over CLIPPER with coverage tools fell from 2.8 to 0.6 points. Including College gives 20.6 ± 0.3 and $22.3 \pm 0.3\%$, an increase of 1.7 ± 0.5 points. The largest changes occur on Water potability ($+10.4$ points) and HELOC ($+3.3$; Figure 7). This change allowed us to standardise behaviour amongst reported numbers.

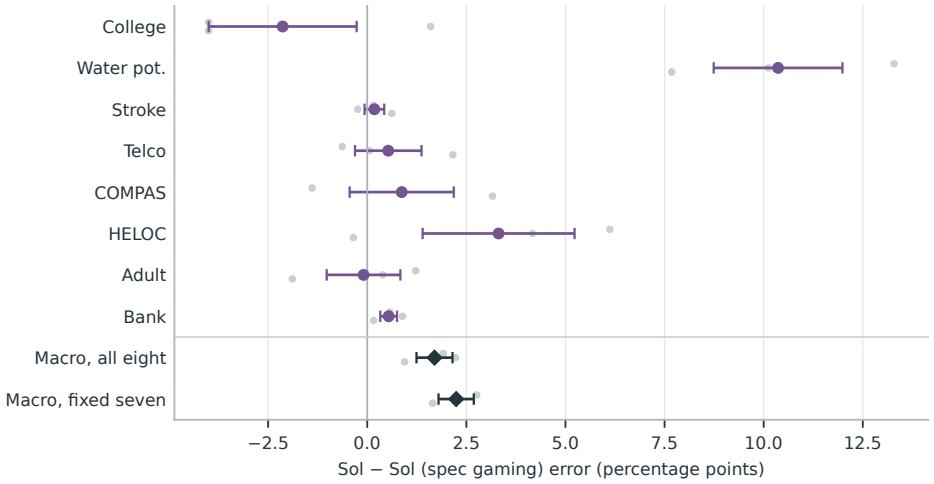


Figure 7: Sol's test-error change when explicitly instructed not to invoke or implement existing learners. Positive values indicate higher error with the restriction. Gray points show three paired seed differences per dataset; colored points and bars show means and SEM. Macro diamonds average datasets equally and propagate the paired SEMs. Both the seven main-text datasets and the complete eight-dataset suite are shown.

The recorded checks found no package-index access or recognized learner-fitting commands in the 24 new runs, and purpose annotations appeared in all 24. The annotations describe profiling class rates and evaluating manually chosen bands and rules. Making the intended procedure explicit therefore substantially narrowed Sol's advantage. This change measures the effect of the instruction, including the strategy it induces. Readable rules alone do not reveal how they were found.

E.4 Anonymization procedure and complete results

Anonymization replaces dataset stems with `task_<hash>`, feature names with `a01`, `a02`, etc., categorical values with `v1`, `v2`, etc., and classes with `y=1` and `y=rest`. The mapping preserves numeric values, missingness, row order, and train/test membership. The original dataset key determines sampling in both conditions, ensuring identical training subsets. Both use Gemini 3.1 Flash-Lite with coverage tools, and all 80 paired fits use the same cross-validated boosting-round budgets. Error increases most on Water potability, Telco churn, COMPAS, and Adult; Stroke and College improve. The eight-dataset macro increase is 1.1 ± 0.2 points (Table 51).

Table 51: Anonymization on all eight datasets at $n = 250$, with ten matched seeds. Entries are mean test error (%) $\pm$ SEM. Changes subtract original from anonymized error, using paired seed differences. Macro rows weight datasets equally and propagate their SEMs.

Dataset	Original	Anonymized	Paired change (pp)
College	19.0 ± 0.4	$\mathbf{16.8 \pm 1.2}$	-2.2 ± 1.1
Water pot.	$\mathbf{38.8 \pm 0.7}$	42.1 ± 0.7	3.3 ± 1.2
Stroke	6.1 ± 0.4	$\mathbf{5.4 \pm 0.3}$	-0.6 ± 0.3
Telco	$\mathbf{22.2 \pm 0.4}$	24.9 ± 0.4	2.7 ± 0.4
COMPAS	$\mathbf{33.5 \pm 0.3}$	36.0 ± 0.4	2.5 ± 0.5
HELOC	$\mathbf{33.1 \pm 0.4}$	33.3 ± 0.5	0.2 ± 0.6
Adult	$\mathbf{17.6 \pm 0.7}$	19.7 ± 0.6	2.2 ± 0.3
Bank	$\mathbf{11.3 \pm 0.2}$	12.4 ± 0.3	1.1 ± 0.3
Macro, all eight	$\mathbf{22.7 \pm 0.2}$	23.8 ± 0.2	1.1 ± 0.2
Macro, fixed seven	$\mathbf{23.2 \pm 0.2}$	24.8 ± 0.2	1.6 ± 0.2

E.5 Backend comparison

Backend comparisons keep the training subsets, $M = 100$ proposal cap, full-training scoring, and boosting-round budgets fixed. A model must return a valid executable predicate as well as find a useful rule: structured-output failures and exhausted tool budgets can consume a round.

Table 52: Complete backend results at $n = 250$, with three matched seeds. Entries are mean test error (%) $\pm$ propagated SEM, averaging datasets equally. The seven main-text datasets exclude College.

Backend	Macro, all eight	Macro, fixed seven
Without coverage tools		
Sonnet 5	$\mathbf{22.8 \pm 0.5}$	$\mathbf{23.3 \pm 0.4}$
Gemini 3.1 Flash-Lite	$\mathbf{22.8 \pm 0.3}$	23.4 ± 0.4
Gemini 3.8	23.7 ± 0.5	24.8 ± 0.6
Gemini 2.5 Flash-Lite	24.1 ± 0.5	23.9 ± 0.3
Qwen 3.5 9B	24.2 ± 0.5	24.8 ± 0.5
Haiku 4.5	24.2 ± 0.4	24.2 ± 0.4
With coverage tools		
Gemini 3.8	$\mathbf{22.6 \pm 0.2}$	23.7 ± 0.2
Gemini 3.1 Flash-Lite	22.9 ± 0.4	$\mathbf{23.5 \pm 0.4}$
Haiku 4.5	23.7 ± 0.4	24.0 ± 0.4
Luna	24.0 ± 0.4	24.6 ± 0.4
Gemini 2.5 Flash-Lite	24.4 ± 0.5	24.8 ± 0.5

F MODEL COMPLEXITY

Table 53 reports the number of retained rules for the six main tabular conditions and Python extraction ensembles. Tabular rule counts vary substantially across fits; for example, hints+tools ranges from 1 to 74 rules at $n = 2{,}000$. For extraction, code lines exclude blank lines, comments, and docstrings, but include imports and function definitions. Tests count each `if`/`elif` and conditional expression once. At $M = 55$, the complete ensemble for one key has a median of 81.5 code lines (interquartile range 61.8–103), summed over its retained rules.

Table 53: Saved model sizes. Tabular cells summarize 70 fits: ten seeds on each of the seven datasets in the main aggregate. Each extraction setting has 100 key ensembles: ten keys and ten seeds. Per-rule summaries pool 635, 625, and 622 retained rules at $M = 20, 55, 110$, respectively. Quartiles use linear interpolation.

(a) Tabular retained rules: mean (median)

Method	$n = 250$	$n = 1{,}200$	$n = 2{,}000$
SLIPPER full	7.0 (3)	7.9 (8)	8.9 (7)
SLIPPER M100	6.1 (3)	6.4 (5.5)	6.2 (3.5)
CLIPPER sample	8.0 (4.5)	9.5 (8)	9.6 (7)
CLIPPER coverage	10.6 (5)	10.8 (9)	11.8 (7)
CLIPPER hints	14.2 (5.5)	15.2 (12)	16.0 (10)
CLIPPER hints+tools	15.3 (6)	16.5 (13.5)	17.4 (10.5)

(b) Python extraction: median [first, third quartile]

Sample cap	Rules per key	Code lines per rule	Tests per rule
$M = 20$	6 [4.8, 9]	11 [11, 14]	3 [2, 3]
$M = 55$	6 [4, 8]	12 [11, 14]	3 [2, 3]
$M = 110$	6 [4.8, 8]	12 [11, 15]	3 [2, 3]

G REPRESENTATIVE PROMPTS AND TOOL TRACES

These examples come from saved CLIPPER coverage-control runs on Adult and Telco churn, using Gemini 3.1 Flash-Lite with 250 training examples and 100 sampled rows per prompt. Prompts are excerpted; each trace includes all coverage calls for that round, in order. The tool returns positive and negative training weights, denoted `p` and `n`.

Shared instructions (excerpts). The system prompt is "You are a concise, careful Python programmer writing a single classification rule." The user prompt includes:

```
Task: write a rule in the form of a Python function
    def covers(x: dict) -> bool | None: ...
that "fires" (returns True) for a meaningful cluster of examples of
class **income=>50K**.  Notes:
 - The rule must be a SINGLE conjunction: exactly one test of the form
       if TEST1 and TEST2 and ... and TESTK: return True
   where each TEST constrains one cluster-relevant attribute.
```

The schema and weighted examples follow. The tool instructions include:

```
  count_coverage_of_lambda_body(body: str) -> (p, n)
    Arbitrary one-line expression in x, wrapped as `lambda x: <body>`.
Budget: **at most 6 tool calls per rule** (hard cap; going over
ends your turn with no rule).  Verify the most promising candidates from the
sample above, then commit.  Reserve at least one call for the final commit.
```

G.1 ADULT: FIRST PROPOSAL

The first two sampled rows are shown below, with selected columns. The trace tests several clusters before adding an age restriction that removes negative weight without losing positive weight. The harness retains the returned rule.

class	weight	age	marital-status	education-num
income=rest	0.5256	26	Never-married	9
income=>50K	1.655	54	Married-civ-spouse	9

Coverage-tool trace. Every call below invokes `count_coverage_of_lambda_body`.

```
Call 1: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('education-num') is not None and
   x.get('education-num') >= 13
  -> p = 0.1578844056706656, n = 0.0263904034896402

Call 2: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('education-num') is not None and
   x.get('education-num') >= 13 and x.get('hours-per-week') is not None and
   x.get('hours-per-week') >= 40
  -> p = 0.14126499454743766, n = 0.02111232279171216

Call 3: body =
x.get('capital-gain') is not None and x.get('capital-gain') > 7000
  -> p = 0.09971646673936775, n = 0.0

Call 4: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('education-num') is not None and
   x.get('education-num') >= 13 and x.get('hours-per-week') is not None and
   x.get('hours-per-week') >= 40 and x.get('age') is not None and x.get('age') >= 30
  -> p = 0.14126499454743766, n = 0.01583424209378412
```

Returned rule.

```
def covers(x: dict) -> bool | None:
    # Rule for income >= 50K: High education, married, and consistent work hours
    if x.get('marital-status') == 'Married-civ-spouse' and \
       x.get('education-num') is not None and x.get('education-num') >= 13 and \
       x.get('hours-per-week') is not None and x.get('hours-per-week') >= 40 and \
       x.get('age') is not None and x.get('age') >= 30:
        return True
    return None
```

G.2 ADULT: A LATER REJECTED PROPOSAL

After two retained rules, the sample and weights have changed. The first two sampled rows now include a more heavily weighted negative example with a bachelor's degree. The agent returns another rule, but it fails the acceptance test, so the harness updates the default instead.

class	weight	age	marital-status	education-num
income=rest	0.5564	19	Never-married	9
income=rest	1.577	31	Married-civ-spouse	13

Coverage-tool trace.

```
Call 1: body =
x.get('capital-gain') is not None and x.get('capital-gain') > 0 and x.get('hours-per-week')
   is not None and x.get('hours-per-week') >= 40
  -> p = 0.06528615401290346, n = 0.032367263701103204

Call 2: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('relationship') == 'Husband' and
   x.get('hours-per-week') is not None and x.get('hours-per-week') >= 45 and x.get('age') is
   not None and x.get('age') >= 35
  -> p = 0.08371510105466755, n = 0.047161983215495834

Call 3: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('education-num') is not None and
   x.get('education-num') >= 13 and x.get('hours-per-week') is not None and
   x.get('hours-per-week') >= 40
  -> p = 0.04501255960941139, n = 0.05667818343799176

Call 4: body =
x.get('marital-status') == 'Married-civ-spouse' and x.get('occupation') == 'Exec-managerial'
   and x.get('hours-per-week') is not None and x.get('hours-per-week') >= 40
  -> p = 0.05530390252446561, n = 0.021867828856022835
```

Returned rule.

```
def covers(x: dict) -> bool | None:
    # Rule for income > 50K: targeting highly committed (40+ hours) married exec-managerial
   staff.
    if x.get('marital-status') == 'Married-civ-spouse' and \
       x.get('occupation') == 'Exec-managerial' and \
       x.get('hours-per-week') is not None and \
       x.get('hours-per-week') >= 40:
        return True
    return None
```

G.3 TELCO CHURN: COMPARING CATEGORICAL CONDITIONS

Here the prompt's target class is `Churn=Yes`. The first two sampled rows are shown below, with selected columns. The agent compares five refinements and returns the electronic-check rule, which the harness retains.

class	weight	InternetService	Contract	PaymentMethod
Churn=Yes	1.401	DSL	One year	Electronic check
Churn=Yes	1.401	Fiber optic	Month-to-month	Electronic check

Coverage-tool trace.

```
Call 1: body =
x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month' and
   x.get('PaperlessBilling') == 'Yes'
  -> p = 0.25610582169871643, n = 0.06333496406126507

Call 2: body =
x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month' and
   x.get('PaymentMethod') == 'Electronic check'
  -> p = 0.2035712941707746, n = 0.043182930041771636

Call 3: body =
x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month' and
   x.get('tenure') is not None and x.get('tenure') < 10
  -> p = 0.11163587099687636, n = 0.01727317201670866

Call 4: body =
x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month' and
   x.get('SeniorCitizen') is not None and x.get('SeniorCitizen') == 1.0
  -> p = 0.13790313476084728, n = 0.02303089602227821

Call 5: body =
x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month' and
   x.get('MonthlyCharges') is not None and x.get('MonthlyCharges') > 80
  -> p = 0.22983855793474553, n = 0.05181951605012596
```

Returned rule.

```
def covers(x: dict) -> bool | None:
    if x.get('InternetService') == 'Fiber optic' and x.get('Contract') == 'Month-to-month'
   and x.get('PaymentMethod') == 'Electronic check':
        return True
    return None
```