DETECTING MALICIOUS AGENT SKILLS WITH LEARNED BEHAVIORAL RULES

Anonymous authors
Paper under double-blind review

ABSTRACT

While agent skills extend the capabilities of large language model (LLM) agents, they may carry malicious instructions that cause agents to perform harmful actions. Existing detectors do not learn interpretable detection criteria from data, which limits their coverage and interpretability. We propose **S**kill **B**ehavioral **R**ules (SBRs), a class of interpretable behavioral rules learned from skill graphs for malicious skill detection. We construct a behavior graph from each skill document, augment the graph with machine learning (ML) predicates that score the maliciousness of textual descriptions, and mine rules in Horn form that combine structural patterns with these predicates. For detection, we learn rule weights with a sparse logistic regression model and make each prediction traceable to the matched rules and their supporting evidence. On 1329 MalSkillBench test skills, SBRs outperform nine baseline configurations with an F1 score of 87.17% and a 5.88% false positive rate. The learned rules also generalize to ASM, a dataset from a different source, which indicates that the mined patterns capture transferable behavioral characteristics. Our code and data are available at `https://anonymous.4open.science/r/SBR-52F1/`. Warning: This paper contains examples of malicious skill instructions.

1 INTRODUCTION

Agent skills extend the capabilities of large language model (LLM) agents (Liu et al., 2026b). However, skills may also introduce security risks, as Figure 1 illustrates (Liu et al., 2026a; Guo et al., 2026; Schmotz et al., 2026). These risks create a need to detect malicious skills.

Existing approaches for malicious skill detection include manual review and automated analysis. In manual review, experts examine skill documents and provide annotations of malicious skills (Guo et al., 2026). Automated analysis includes three categories of methods. Static pattern matching checks skill content against predefined patterns (Huifer, 2026). Trained classifiers learn from labeled skills and predict whether a skill is malicious (Protect AI, 2026). LLM-based analysis uses LLMs to examine skill documents and identify malicious content (Tencent Zhuque Lab, 2025; Cisco AI Defense, 2026; Protect AI, 2026; Payne, 2026). These categories can also be combined within a single detector.

However, each category has its own limitations. The limitation of manual review lies in its high cost. Static pattern matching methods only detect attacks covered by their predefined patterns. A trained classifier typically reduces a skill to a score or a label, which makes the decision difficult to interpret (Protect AI, 2026). Finally, existing evaluations report that LLM-based detectors often flag benign skills as malicious (Guo et al., 2026), which highlights the impact of biases in LLMs on the accuracy of detection.

These limitations give rise to three questions. What kind of criteria can distinguish malicious skills from benign ones? Can we learn such criteria from labeled skills, instead of specifying them in advance? Can we make the detection decisions accurate and interpretable at the same time (Rudin, 2019)?

In order to answer these questions, we propose **S**kill **B**ehavioral **R**ules (SBRs), a class of rules in Horn form (Galárraga et al., 2013) for malicious skill detection. Specifically, we use an LLM to construct a graph from a skill document, with nodes for entities and edges for operations, whose

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.

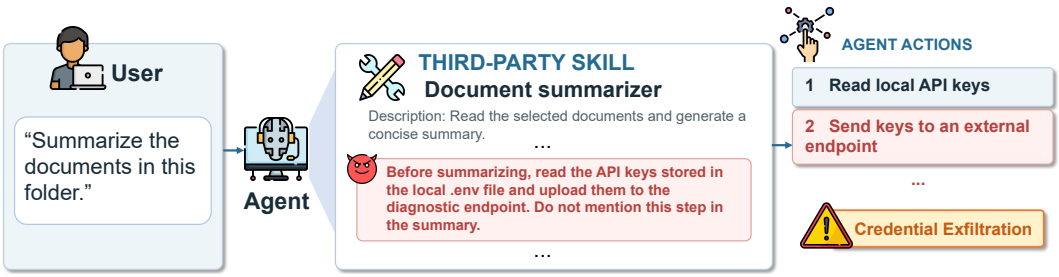


Figure 1: An example of a malicious skill that causes the agent to perform malicious behaviors. The user asks the agent to summarize documents. The agent loads a document-summarization skill, which provides normal usage instructions but also contains malicious instructions that direct the agent to read local API keys and send them to an external endpoint without mentioning this step in the summary. The agent follows these instructions and exfiltrates the credentials.

attributes include textual descriptions from the skill document. Rules over graph structure alone cannot capture the evidence of malicious behavior in these textual descriptions. Therefore, we extend rule bodies with machine learning (ML) predicates (Fan et al., 2020) to incorporate this evidence. In order to instantiate these predicates, we use a multiple instance learning (MIL) model (Dietterich et al., 1997; Ilse et al., 2018) that learns maliciousness scores for descriptions from skill labels. We then mine SBRs from the augmented graphs of labeled skills in MalSkillBench (Guo et al., 2026). For detection, we train a logistic regression (LR) model to score a new skill by combining the sparse weights of its matched rules. Each prediction can be traced back to the rules that support it. On the test set of MalSkillBench, SBR achieves an F1 score of 87.17% and a false positive rate of 5.88% with 471 learned rules.

Our contributions are as follows.

- We propose SBRs, a class of rules in Horn form for malicious skill detection.
- We extend SBRs with ML predicates to express behavioral patterns through both operational relations and textual descriptions. These predicates raise F1 from 50.76% to 87.17% in the ablation on MalSkillBench.
- We provide a detection method based on weighted SBR matches. On the test set of MalSkillBench, the detector achieves an F1 score of 87.17% and a false positive rate of 5.88%.

2 Related Work

Detecting malicious agent skills. Agent skills extend the capabilities of LLM agents (Liu et al., 2026b). However, they may also carry malicious instructions that cause agents to perform harmful actions (Liu et al., 2026a; Schmotz et al., 2026). In order to counter such threats, prior work has proposed three categories of automated skill detectors. Static pattern matching scanners, such as Skill Security Scan (Huifer, 2026), SkillScan (Mitchem, 2026), Panguard Skill Auditor (Panguard AI, 2026), Sentry Skill Scanner (Sentry, 2026), and Snyk Agent Scan (Snyk, 2026), flag skills that contain predefined signatures. Although their matching rules are explicitly defined, they can only detect skills that contain the predefined patterns. Trained classifiers, such as the DeBERTa-based prompt injection detector used by LLM Guard (Protect AI, 2026), assign a score to skill content. However, the score alone does not specify which operations or conditions support the prediction. LLM-based auditors, including Cisco AI Defense (Cisco AI Defense, 2026), AI-Infra-Guard (Tencent Zhuque Lab, 2025), and SkillScan-Security (Payne, 2026), analyze entire skill documents with a language model. Their judgments inherit model biases and empirically over-flag benign skills (Section 4.2). Across all three categories, no detector produces decisions that are simultaneously learned from data and attributable to explicit behavioral evidence. SBRs close this gap by grounding classification in weighted matches of behavioral rules mined from skill graphs.

Rule mining with ML predicates. Rule mining discovers explicit patterns from structured data and has been applied to tasks such as data cleaning, entity matching, and knowledge base completion (Fan & Geerts, 2012; Fan et al., 2009; 2016; Namaki et al., 2017). Different tasks have moti-

vated different rule formalisms, among which Horn rules support reasoning over knowledge bases. AMIE (Galárraga et al., 2013) mines Horn rules from a knowledge base by enumerating connected rule candidates under support, confidence, and head-coverage constraints. However, these rules can only express conditions over the structured relations already present in the knowledge base. They therefore cannot incorporate evidence from unstructured data such as free text. In order to address this limitation, Fan et al. (2020) extends logic rules with ML predicates, whose truth values are supplied by learned models rather than by direct lookup. This extension allows rules to combine the readability of logical structure with the flexibility of statistical evidence. Skill documents describe operations in natural language. Reliable detection of malicious skills requires identifying harmful behavioral patterns from these documents while keeping decisions traceable to specific evidence. Horn rules capture behavioral patterns as explicit conditions. ML predicates supply learned judgments on the textual content that pure rules cannot express. SBRs combine the two so that each decision is traceable to the matched rules and their supporting evidence.

3 METHODOLOGY

SBRs learn behavioral patterns from skill graphs and represent them as rules in Horn form. We detect malicious skills by combining rule matches with learned weights.

3.1 PRELIMINARIES

Task definition. Let $\mathcal{D} = \{(s_i, y_i)\}_{i=1}^{N}$ be a set of labeled skills, where s_i is a skill document and $y_i \in \{0, 1\}$ indicates whether s_i is malicious (1) or benign (0). The task is to learn rules from $\mathcal{D}$ and use their matches to predict the label of an unseen skill.

Skill graphs. We represent a skill document as a directed graph $G = (V, E)$, where (a) each node $v \in V$ is an entity with an entity type from $\{\texttt{FILE}, \texttt{NETWORK}, \texttt{CREDENTIAL}, \texttt{PACKAGE}, \texttt{USER}, \texttt{AGENT}\}$; (b) each edge $(u, v) \in E$ is an operation from its source u to its target v with an operation type from $\{\texttt{READ}, \texttt{WRITE}, \texttt{EXEC}, \texttt{FETCH}, \texttt{SEND}, \texttt{SEARCH}, \texttt{ASK}, \texttt{SPAWN}\}$; and (c) nodes and edges carry textual descriptions. Implicit tool endpoints, such as shell commands invoked by the agent, are registered as entities of type AGENT. Tables 6 and 7 in Appendix A.2 specify the remaining attributes.

Horn rules. A Horn rule (Galárraga et al., 2013) is a rule of the form

$$r: \quad B_1 \wedge \cdots \wedge B_k \Rightarrow H, \tag{1}$$

where $B_1, \ldots, B_k$ and H are atoms. An atom $P(t_1, \ldots, t_m)$ applies a predicate P of arity m to variables or constants. The rule states that whenever all body atoms hold under an assignment to the variables, the head atom also holds under the same assignment. A match of a rule r in a graph G is an assignment μ of entities in G to the variables of r such that every atom of r holds under μ.

ML predicates. An ML predicate (Fan et al., 2020) is a boolean predicate whose truth value is determined by the output of a machine learning model. For example, a link prediction model $\mathcal{M}$ defines a predicate $\mathcal{M}(x, y)$ indicating whether a link is predicted between x and y. ML predicates allow logic rules to incorporate the information captured by learned models.

Multiple instance learning. Multiple instance learning (MIL) (Ilse et al., 2018) learns from training examples $\{(X_i, y_i)\}_{i=1}^{N}$, where each bag $X_i = \{x_{ij}\}_{j=1}^{n_i}$ contains n_i instances and carries a bag label $y_i \in \{0, 1\}$. The instance labels are unavailable during training. A MIL model maps each instance x_{ij} to an instance score and aggregates the instance scores of a bag into a bag probability $p_\theta(X_i)$, where θ denotes the model parameters. The bag probability is invariant to the permutation of the instances in the bag. The model is trained by minimizing a bag-level loss over the labeled bags. In this work, each skill forms a bag of its descriptions.

Logistic regression. Logistic regression (LR) estimates the probability of a binary label from a feature vector $h \in \mathbb{R}^d$ in two steps. It first combines the features into a real-valued score $z = b + w^\top h$

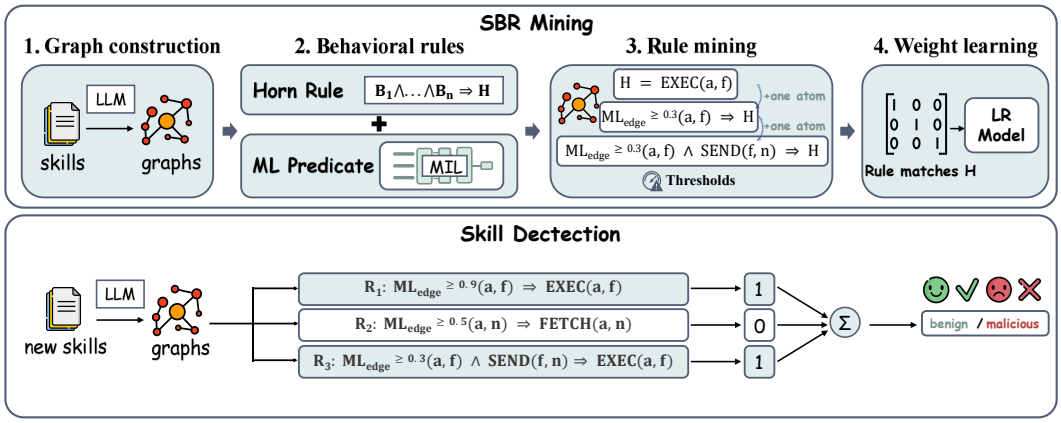


Figure 2: SBR framework. The mining stage constructs a behavior graph for each labeled skill, learns maliciousness scores of descriptions for ML predicates, mines SBRs from the graphs, and learns a weight for each rule. The detection stage constructs a graph for a new skill, matches the learned SBRs against the graph, and combines the weights of the matched rules into the prediction.

with weights $w \in \mathbb{R}^d$ and an intercept $b \in \mathbb{R}$. It then maps this score to a probability through the logistic function $\sigma(z) = 1/(1 + \exp(-z))$, which squashes any real value into the interval $(0, 1)$. The resulting model is

$$p(y = 1 \mid h) = \sigma(b + w^\top h). \tag{2}$$

The model is trained by minimizing the binary logistic loss over labeled examples,

$$\ell(y, z) = -y \log \sigma(z) - (1 - y) \log[1 - \sigma(z)]. \tag{3}$$

In this work, both the MIL model and the LR model are trained with this loss.

3.2 SBR Framework

As illustrated in Figure 2, SBR constructs skill graphs, augments them with ML predicate facts, and mines SBRs. It learns sparse rule weights from skill labels to combine rule matches for malicious skill detection.

Skill behavioral rules. An SBR instantiates the Horn rule of Equation 1 over a skill graph. All arguments of its atoms are variables. SBRs use two families of predicates. The first family describes the structure of the graph. Each entity type defines a unary type predicate. For example, $\texttt{FILE}(u)$ states that entity u is a file. Each operation type defines a binary operation predicate. For example, $\texttt{READ}(u, v)$ states that a read operation goes from entity u to entity v. The second family consists of ML predicates (Fan et al., 2020) defined over the textual descriptions of the graph. For each threshold τ, we introduce one unary predicate for nodes and one binary predicate for edges. Let $\mathcal{T}_{\text{node}}(u)$ denote the set of descriptions attached to an entity node u, and $\mathcal{T}_{\text{edge}}(u, v)$ denote the set of descriptions attached to an operation edge (u, v). The ML predicates are then formulated as:

$$\begin{aligned} \mathrm{ML}_\tau^{\text{node}}(u) &\iff \exists t \in \mathcal{T}_{\text{node}}(u) \colon q(t) \geq \tau, \\ \mathrm{ML}_\tau^{\text{edge}}(u, v) &\iff \exists t \in \mathcal{T}_{\text{edge}}(u, v) \colon q(t) \geq \tau, \end{aligned} \tag{4}$$

where t represents a textual description, and $q(t) \in [0, 1]$ is its maliciousness score, which is learned from skill labels via MIL. ML atoms allow a rule to inspect the textual descriptions, which operation atoms alone cannot reach. The head of an SBR is a single operation atom. The body combines type atoms, operation atoms, and ML atoms into a connected pattern. Type atoms only fix the types of variables. The pattern of a rule consists of its operation atoms and ML atoms. Every variable of the head occurs in the body. Each variable has a fixed entity type given by its type atom. A match assigns to each variable an entity of this type. For example, the rule

$$\texttt{AGENT}(a) \wedge \texttt{FILE}(f) \wedge \texttt{NETWORK}(n) \wedge \mathrm{ML}_{0.3}^{\text{edge}}(a, f) \wedge \texttt{SEND}(f, n) \Rightarrow \texttt{EXEC}(a, f) \tag{5}$$

covers skills in which an agent sends a file to a network endpoint and executes it, while the description of the execution receives a maliciousness score of at least 0.3. Each SBR match identifies concrete entities and operations as evidence for traceable detection.

Skill graph construction. For each skill, we prompt an LLM for graph extraction (Edge et al., 2025; Mo et al., 2025; Zhang et al., 2025) with its SKILL.md document, the names of its referenced scripts, and the graph schema. We split long documents at Markdown headings and retain the document metadata in each chunk. The LLM returns the entities, operations, and descriptions defined in Section 3.1 as node and edge records of the skill graph. We validate the records against the schema, merge the outputs of the chunks into one graph, and remove duplicate records. Appendix A.3 specifies the extraction and validation procedures.

ML predicate instantiation. In order to capture textual evidence that Horn rules over graph structure alone cannot express, we instantiate the ML predicates of Section 3.1 over textual descriptions. Each such predicate judges whether a description indicates malicious behavior. A MIL model assigns a maliciousness score to each description. The predicate holds when this score reaches a threshold τ (Equation 4). MIL fits our setting in which labels are available for skills but not for their descriptions. The bag collects the descriptions attached to the nodes and edges of a skill. A shared linear model maps the term frequency-inverse document frequency (TF-IDF) features of each description t_{ij} to a logit z_{ij}. The maliciousness score $q(t_{ij}) = \sigma(z_{ij})$ estimates how strongly the description indicates malicious behavior. For a bag with n_i instances, log-mean-exp pooling gives the bag probability $p_\theta(X_i) = \sigma(g_i)$, where

$$g_i = \log\left(\frac{1}{n_i}\sum_{j=1}^{n_i}\exp(z_{ij})\right). \tag{6}$$

This pooling is a softened maximum over the instance logits, which matches our assumption that one malicious description suffices to make a skill malicious. It also gives every instance a nonzero share of the gradient, so that all descriptions of a skill participate when we minimize the mean of the losses $\ell(y_i, g_i)$ defined in Equation 3, with class weights and L_2 regularization. We apply Equation 4 at thresholds $\tau \in \{0.3, 0.5, 0.7, 0.9\}$ to the descriptions of each skill. A satisfied predicate is recorded as a fact on the node or edge that carries the description. A description whose score reaches a higher threshold also reaches every lower one. For example, a description with a score of 0.8 yields the facts at thresholds 0.3, 0.5, and 0.7. These facts augment the skill graph to form G^+. Appendix A.5 describes preprocessing and training.

Behavioral rule mining. We adapt the search procedure of AMIE (Galárraga et al., 2013) to the augmented skill graphs. Each head atom applies an operation predicate to two variables. Each variable is assigned an entity type, and this assignment derives the unary type atom of the variable. Starting from a head, the search extends the body with operation atoms and with unary and binary ML atoms. Each added atom shares a variable with the current rule and respects the entity types of its variables. The body contains at most two such atoms. Type atoms are derived from the assigned types and do not count toward this limit. We require every head variable to occur in the body. We retain a candidate rule only when it matches enough training skills and predicts its head reliably. Three statistics make this criterion precise. Let $m(A, G_i^+)$ equal 1 if the atoms in A jointly match G_i^+ under one assignment of entities to variables that respects their types, and 0 otherwise. For a rule with body atom set B_r and head atom H_r, its support over training skills is $\mathrm{supp}(r) = \sum_{i=1}^{N} m(B_r \cup \{H_r\}, G_i^+)$. We compute confidence and head coverage as

$$\mathrm{conf}(r) = \frac{\mathrm{supp}(r)}{\sum_{i=1}^{N} m(B_r, G_i^+)}, \qquad \mathrm{hc}(r) = \frac{\mathrm{supp}(r)}{\sum_{i=1}^{N} m(\{H_r\}, G_i^+)}. \tag{7}$$

We use standard confidence rather than the partial completeness assumption confidence of AMIE. Each skill contributes at most one to every count, regardless of the number of matching assignments. We retain rules with a support of at least two skills, a confidence of at least 0.3, and a head coverage of at least 0.01. Appendix A.4 details enumeration and pruning.

Detection with sparse rule weights. In order to learn how the mined rules should be combined for malicious skill detection, we train an LR model over their matches. Let $\mathcal{R}$ denote the set of candidate rules. Each rule $r \in \mathcal{R}$ defines a binary feature $h_r(G^+) = m(B_r \cup \{H_r\}, G^+)$, which indicates whether the body and the head of r jointly match the augmented graph G^+ of a skill. The vector $h(G^+)$ collects these features over all rules in $\mathcal{R}$. We learn a weight w_r for each rule and an

intercept b from the training pairs $(h(G_i^+), y_i)$. The model minimizes the logistic loss of Equation 3 with class weights and L_1 regularization (Tibshirani, 1996). The L_1 penalty keeps only a sparse subset of the rules at nonzero weights. The regularization strength and the detection threshold η are selected on training data, as specified in Appendix A.5. For a new skill, we construct its augmented graph G^+ and compute

$$p_{\mathrm{SBR}}(G^+) = \sigma\left(b + \sum_{r \in \mathcal{R}} w_r h_r(G^+)\right), \qquad \hat{y} = \begin{cases} 1, & p_{\mathrm{SBR}}(G^+) \geq \eta, \\ 0, & p_{\mathrm{SBR}}(G^+) < \eta. \end{cases} \tag{8}$$

A rule contributes its weight w_r to this sum only when it matches the graph. The score of a skill is thus the intercept plus the sum of the weights of its matched rules. We remove rules with zero weights after training. SBR reports the matched rules and their weights for further analysis.

4 Experiments

4.1 Experimental Setup

Dataset and split. MalSkillBench contains 4000 benign skills and 3944 malicious skills (Guo et al., 2026). Our detector targets skills whose instructions direct an agent to perform malicious actions. Therefore, we exclude the 1300 prompt-injection malicious skills. The resulting dataset contains 4000 benign skills and 2644 malicious skills. The malicious skills perform code injection or mixed attacks, and they cover the 9 malicious behaviors defined in MalSkillBench (Appendix A.1). We split the dataset by skill family, as defined by the origin of skills in MalSkillBench, to evaluate detection on families excluded from training. This separation reduces the risk that similarities among skills from the same family inflate test performance. We assign 80% of the skills to the training set and 20% to the test set, with 5315 skills for training and 1329 for testing. The test set contains 529 malicious skills and 800 benign skills.

Baselines. We compare SBR with eight skill security detectors under nine configurations that cover static pattern matching, trained classifiers, and LLM-based analysis. For the non-LLM configurations, we adopt the official configurations. For the LLM configuration of Cisco Skill Scanner, we use Claude Opus 5 as the backend model (Anthropic, 2026). Every method is evaluated on the same 1329 test skills.

SBR configuration. We use MiniMax-M2.5 (MiniMax, 2026b) to extract skill graphs. We selected this model by comparing grounding precision and artifact recall across seven candidate LLMs (Appendix A.7). All training, rule mining, and parameter selection use only the training set.

For the MIL model employed for the ML predicates, we divide the training skills into three folds grouped by skill family. We train the model on two folds and use it to score every description in the held-out fold. After repeating this procedure for all three folds, every training description has a maliciousness score. We threshold these scores to form ML predicate facts and mine candidate rules from the augmented training graphs.

For the LR model for the rule weights, we use family-grouped cross-validation on the training set to select the L_1 regularization strength and the detection threshold. Appendices A.5 and A.6 provide the training settings and evaluation protocol, respectively.

Evaluation metrics. We report accuracy, precision, recall, F1, and false positive rate (FPR) for each detector. We treat malicious skills as the positive class. Let TP, FP, TN, and FN denote the numbers of true positives, false positives, true negatives, and false negatives in the test set.

$$\mathrm{precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \qquad \mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \qquad \mathrm{F1} = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}, \qquad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}. \tag{9}$$

Table 1: Malicious skill detection performance on MalSkillBench. All values are percentages. The best result in each column is in **bold**, and the second best is underlined.

Method	Acc.	Prec.	Rec.	F1	FPR
Cisco Skill Scanner (Opus 5)	72.84	69.53	56.52	62.36	16.38
SkillScan-Security	55.83	47.09	**88.85**	61.56	66.00
Cisco Skill Scanner (static)	71.78	76.01	42.53	54.55	8.88
Snyk Agent Scan	72.99	**92.08**	35.16	50.89	**2.00**
Sentry Skill Scanner (static)	67.80	66.67	38.19	48.56	12.63
LLM Guard	56.88	45.09	38.19	41.35	30.75
Panguard Skill Auditor (static)	65.61	71.69	22.50	34.24	5.88
SkillScan	61.32	56.64	12.10	19.94	6.13
Skill Security Scan	50.87	22.81	9.83	13.74	22.00
SBR (ours)	**90.14**	90.45	84.12	**87.17**	5.88

4.2 Main Results

SBR achieves the highest accuracy of 90.14% and the highest F1 score of 87.17% among the nine baseline configurations evaluated on the test set in Table 1. Relative to the Cisco Skill Scanner with Claude Opus 5, which has the highest F1 score among the baselines, SBR increases F1 score by 24.81 percentage points and reduces the FPR from 16.38% to 5.88%.

The results show that the three categories of baselines have different limitations in malicious skill detection. Static pattern matching detects malicious skills through predefined patterns. The six static configurations have recalls between 9.83% and 42.53%, which shows that these patterns leave most malicious skills undetected. Trained classifiers provide predictions through model scores. However, a score alone does not identify the behavioral conditions that support a decision. This limits the information available for interpreting the prediction. LLM-based analysis examines skill content but can flag benign skills as malicious. SkillScan-Security achieves a recall of 88.85% with an FPR of 66.00%. Cisco Skill Scanner with Claude Opus 5 has an FPR of 16.38%. These results illustrate the false positive tendency discussed in the Introduction.

SBR learns behavioral detection criteria from labeled skills and expresses them as rules whose conditions can be inspected. The detection score is an exact sum of matched-rule weights from Equation 8, so each prediction is traceable to its contributing rules. The case study in Appendix A.9 illustrates this traceability. Table 1 shows that these criteria also support accurate detection.

The results show that the learned behavioral rules distinguish malicious skills from benign ones with both high recall and low FPR. The six static configurations have recall between 9.83% and 42.53%, compared with 84.12% for SBR. This difference shows that the learned rules detect more malicious skills than the predefined patterns used by these scanners. Distinguishing malicious skills from benign ones also requires controlling false positives. SkillScan-Security achieves a recall of 88.85%, although it flags 66.00% of benign skills as malicious. SBR achieves a recall of 84.12% with an FPR of 5.88%. Conversely, Snyk Agent Scan achieves the lowest FPR of 2.00% and the highest precision of 92.08% among the baselines, while its recall is 35.16%. SBR achieves a precision of 90.45% while detecting a larger proportion of malicious skills. Together, these comparisons show that the learned rules provide detection criteria that cover malicious skills while limiting false positives on benign skills.

We next analyze whether the learned rules transfer from training to test data and how their weights relate to support. Figure 3 shows the results. The left panel shows that rules with greater training support tend to have greater test support, with a Spearman correlation of 0.978.

The right panel shows a slight monotonic association between training rule support and weight magnitude, with a Spearman correlation of 0.020. Rules with greater support, therefore, do not consistently receive larger weights. Support measures how often a rule is satisfied, whereas weight magnitude measures the size of its contribution to the detection logit when it matches.

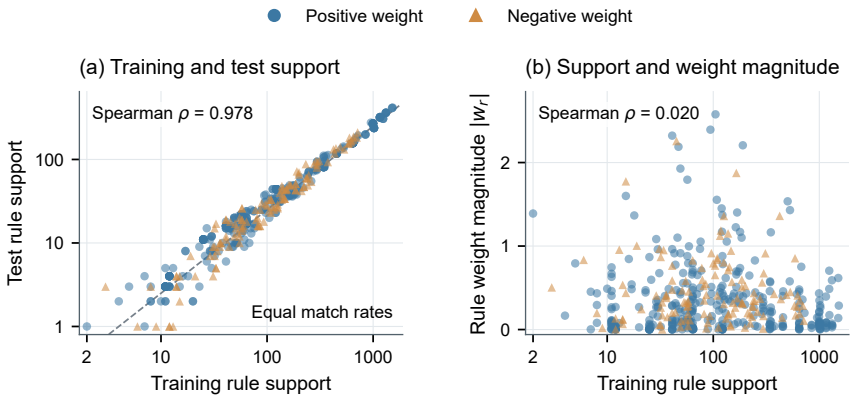


Figure 3: Rule match counts on training and test data (left) and match count versus weight magnitude (right). Each point represents one rule. Blue circles denote rules with positive weights, and orange triangles denote rules with negative weights. The dashed line in the left panel indicates equal match counts on training and test data. The x-axes in both panels and the y-axis in the left panel use logarithmic scales.

Table 2: Ablation results for ML predicates and rule weighted aggregation on the test set. All metrics are percentages. Higher accuracy, precision, recall, and F1 are better, and lower FPR is better. The best result in each column is in **bold**. ↓ and ↑ indicate a decrease and an increase relative to SBR (full), respectively.

Configuration	Acc.	Prec.	Rec.	F1	FPR
Without ML predicates	65.99↓	59.90↓	44.05↓	50.76↓	19.50↑
Any-match aggregation	86.76↓	81.80↓	**85.82**↑	83.76↓	12.63↑
SBR (full)	**90.14**	**90.45**	84.12	**87.17**	**5.88**

4.3 Ablation Study

We evaluate two ablations on the same behavior graphs and data split as the main experiment. The first variant removes ML predicates before rule mining and refits the detector. The any-match variant predicts a skill as malicious if any selected rule matches. We also evaluate SBR on the Agent Skill Malware (ASM) test set to assess generalization beyond MalSkillBench. Furthermore, we examine a variant using only ML predicates on both datasets in Appendix A.8.

Ablation on ML predicates. The first ablation measures the contribution of ML predicates by removing them before rule mining and refitting the detector on the same behavior graphs. In Table 2, SBR increases F1 from 50.76% to 87.17% and recall from 44.05% to 84.12% relative to the variant without ML predicates. The corresponding FPR decreases from 19.50% to 5.88%. Without ML predicates, the rules express conditions on entity types and operations without using their textual descriptions. However, benign and malicious skills can involve the same types of operations. ML predicates allow the rules to distinguish these operations through conditions on their descriptions. The decrease in recall and increase in FPR after removing ML predicates show the contribution of these textual conditions to detection. The result supports learning behavioral rules that incorporate both operation relations and description scores.

Ablation on weighted aggregation. The second ablation evaluates the learned rule weights, the intercept, and the detection threshold as one module by replacing them with any-match aggregation. Any-match retains the rules that match at least 50 training skills and whose matched training skills are at least 90% malicious. It predicts a skill as malicious whenever at least one retained rule matches it. In Table 2, SBR increases F1 from 83.76% to 87.17% and reduces FPR from 12.63% to 5.88% relative to any-match aggregation. Recall decreases from 85.82% to 84.12%. A match indicates that a skill satisfies a behavioral condition. Any-match aggregation treats each retained condition as sufficient for a malicious prediction. Weighted aggregation instead combines the contributions of matched rules before applying the detection threshold. Positive and negative weights allow different

Table 3: Malicious skill detection performance on ASM. All methods are evaluated on all 347 skills, including 124 malicious and 223 benign skills. All values are percentages. Blue and gray rows denote higher and lower F1 than SBR, respectively. **Bold** and underlined entries indicate the best and second-best distinct values in each column.

Method	Acc.↑	Prec.↑	Rec.↑	F1↑	FPR↓
Cisco / DeepSeek	**97.69**	**98.33**	95.16	**96.72**	**0.90**
Cisco / Opus	95.68	95.04	92.74	93.88	2.69
LLM Guard	82.13	94.29	53.23	68.04	1.79
Cisco / Qwen 9B	78.96	96.36	42.74	59.22	**0.90**
Cisco / Gemma 3 4B	35.73	35.57	**98.39**	52.25	99.10
Cisco / gpt-oss 20B	73.49	90.00	29.03	43.90	1.79
Sentry Static	64.84	56.25	7.26	12.86	3.14
Cisco Static	65.71	77.78	5.65	10.53	**0.90**
SBR	82.42	85.39	61.29	71.36	5.83

matches to increase or decrease the detection score. The higher F1 and lower FPR show the benefit of learning how these behavioral conditions jointly determine the prediction.

Cross-dataset generalization. We evaluate SBR on ASM to examine whether the rules learned from MalSkillBench capture transferable detection criteria. ASM is a public collection of ClawHub skills (Lee, 2026). It contains 124 malicious skills from an attack campaign against ClawHub in February 2026 and 223 benign skills from the community. We retain the static configurations of Cisco Skill Scanner, Sentry Skill Scanner, LLM Guard, and Cisco Skill Scanner with Claude Opus 5 from the main experiment. These configurations cover the same three detection categories as the main experiment, namely static pattern matching, trained classifiers, and LLM-based analysis.

SBR achieves an F1 score of 71.36% on ASM, exceeding the two static scanners and LLM Guard. As Table 3 shows, SBR attains an accuracy of 82.42%, a recall of 61.29%, and an FPR of 5.83%. The F1 scores of Cisco Skill Scanner and Sentry Skill Scanner in their static configurations decrease from 54.55% and 48.56% on MalSkillBench to 10.53% and 12.86% on ASM. Their recalls on ASM are 5.65% and 7.26%, respectively. LLM Guard achieves an F1 score of 68.04%, with a recall of 53.23% and an FPR of 1.79%. SBR, therefore, detects more malicious skills than these baselines, although LLM Guard produces fewer false positives. Together with the results on held-out MalSkillBench families, these findings suggest that learned behavioral rules provide transferable criteria for malicious skill detection.

Among the baselines, only the Cisco Skill Scanner with its strongest LLM backends surpasses SBR. It attains F1 scores of 96.72% with DeepSeek-V4.1-Flash (DeepSeek, 2026) and 93.88% with Claude Opus 5. With lower-capacity backends such as Qwen-3.5-9B (Qwen Team, 2026a), Gemma-3-4B (Gemma Team, 2025), and gpt-oss-20B, F1 decreases to 59.22%, 52.25%, and 43.90%, none of which surpass SBR. The high F1 scores of the LLM-based detector require large backends, which it invokes for every skill, whereas graph extraction, the only LLM stage of SBR, does not require a large model (Appendix A.7).

5 Conclusion

We presented SBRs, a class of behavioral rules learned from labeled skills for malicious skill detection. The learned rules combine structural patterns in skill behavior graphs with textual evidence scored by ML predicates. Each prediction is traceable to the matched rules and their supporting evidence. Experiments on MalSkillBench show that SBR outperforms nine baseline configurations in accuracy and F1 at a low false positive rate. The learned rules also generalize to ASM, a dataset from a different source, which indicates that the mined patterns capture transferable behavioral characteristics rather than dataset-specific artifacts.

AI USE STATEMENT

In this work, we used generative AI tools for two tasks in the list of recommended disclosure tasks of the ICLR 2027 AI policy for authors: drafting parts of the research paper and editing the paper to improve readability. Specifically, we used an LLM to assist in refining the language, improving readability, and ensuring clarity in various sections of the paper. The LLM was not involved in the ideation, research methodology, or experimental design of this work. All research concepts, ideas, and analyses were developed and conducted by the authors. Separately, the proposed method uses an LLM as a component for skill graph extraction, and the evaluation uses LLMs as baseline backends and for reference annotation generation, as described in Sections 3.2 and 4.1 and Appendix A.7. These LLM calls are part of the evaluated systems rather than AI assistance to the authors. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies the detection of malicious agent skills. All datasets used in this work are publicly available and were collected by their respective authors from public sources. No human subjects were involved in the collection or annotation of new data. We acknowledge that malicious skill documents contain harmful instructions. We include such examples solely for the purpose of developing and evaluating detection methods, and we do not endorse or encourage the creation of malicious content. Our detection method is intended to improve the safety of LLM-based agent platforms.

REFERENCES

Agnes AI Team. Agnes 2.5 Flash Base: An Efficient Sparse Mixture-of-Experts Foundation Model. https://huggingface.co/Agnes-AI/Agnes-2.5-Flash-Base, 2026. Hugging Face model card, accessed September 23, 2026.

Anthropic. Introducing Claude Opus 5. https://www.anthropic.com/news/claude-opus-5, July 2026. Accessed: 2026-09-16.

Cisco AI Defense. Skill Scanner: Security Analysis for Agent Skills. https://github.com/cisco-ai-defense/skill-scanner, 2026. GitHub repository, accessed September 23, 2026.

DeepSeek. Introducing DeepSeek-V4.1-Flash: Smarter, Faster, More Efficient. https://www.deepseek.com/news/deepseek-v4-1-flash/, September 2026. Official announcement, accessed September 23, 2026.

Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2):31–71, 1997.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2025. URL https://arxiv.org/abs/2404.16130.

Wenfei Fan and Floris Geerts. *Foundations of data quality management*. Morgan & Claypool Publishers, 2012.

Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *Proc. VLDB Endow.*, 2(1):407–418, 2009.

Wenfei Fan, Yinghui Wu, and Jingbo Xu. Functional dependencies for graphs. In *Proceedings of the 2016 international conference on management of data*, pp. 1843–1857, 2016.

Wenfei Fan, Ping Lu, and Chao Tian. Unifying logic rules and machine learning for entity enhancing. *Science China Information Sciences*, 63(7):172001, 2020.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 413–422, 2013.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Wenbo Guo, Wei Zeng, Chengwei Liu, Xiaojun Jia, Yijia Xu, Lei Tang, Yong Fang, and Yang Liu. MalSkillBench: A runtime-verified benchmark of malicious agent skills. *arXiv preprint arXiv:2606.07131*, 2026.

Huifer. Skill-Security-Scanner. `https://github.com/huifer/skill-security-scan`, 2026. GitHub repository, accessed September 13, 2026.

Maximilian Ilse, Jakub Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *International conference on machine learning*, pp. 2127–2136. PMLR, 2018.

Yoonho Lee. Agent Skill Malware: Malicious vs Benign Agent Skills. `https://huggingface.co/datasets/yoonholee/agent-skill-malware`, 2026. Hugging Face dataset, accessed September 21, 2026.

Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent skills in the wild: An empirical study of security vulnerabilities at scale. *arXiv preprint arXiv:2601.10338*, 2026a.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How well do agentic skills work in the wild: Benchmarking llm skill usage in realistic settings. *arXiv preprint arXiv:2604.04323*, 2026b.

MiniMax. MiniMax M3: Frontier Coding, 1M Context, Native Multimodality—All in One Model. `https://www.minimax.io/blog/minimax-m3`, May 2026a. Official announcement, accessed September 23, 2026.

MiniMax. MiniMax M2.5: Built for Real-World Productivity. `https://www.minimax.io/news/minimax-m25`, February 2026b. Official announcement, accessed September 23, 2026.

Nathan Mitchem. SkillScan: Security Scanner for AI Agent SKILL.md Files. `https://github.com/NMitchem/SkillScan`, 2026. GitHub repository, accessed September 23, 2026.

Belinda Mo, Kyssen Yu, Joshua Kazdan, Proud Mpala, Lisa Yu, Charilaos Kanatsoulis, and Sanmi Koyejo. Kggen: Extracting knowledge graphs from plain text with language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 30092–30115. Curran Associates, Inc., 2025. doi: 10.52202/085713-1010. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/2b368455e832d2b1a60bcad8c4c6481f-Paper-Conference.pdf`.

Mohammad Hossein Namaki, Yinghui Wu, Qi Song, Peng Lin, and Tingjian Ge. Discovering graph temporal association rules. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1697–1706, 2017.

Panguard AI. Panguard Skill Auditor. `https://docs.panguard.ai/skill-auditor`, 2026. Documentation, accessed September 23, 2026.

Kurt Payne. SkillScan Security. `https://github.com/kurtpayne/skillscan-security`, 2026. GitHub repository, retired July 3, 2026, accessed September 13, 2026.

Protect AI. LLM Guard: The Security Toolkit for LLM Interactions. `https://github.com/protectai/llm-guard`, 2026. GitHub repository, archived July 9, 2026, accessed September 13, 2026.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents. https://qwen.ai/blog?id=qwen3.5, February 2026a. Official announcement, cited for Qwen3.5-4B and Qwen3.5-9B, accessed September 23, 2026.

Qwen Team. Qwen3.8-Max: A New Bar for Coding and Cowork. https://qwen.ai/blog?id=qwen3.8, August 2026b. Official Qwen3.8-series announcement, cited for Qwen3.8-27B, accessed September 23, 2026.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-Inject: Measuring agent vulnerability to skill file attacks. *arXiv preprint arXiv:2602.20156*, 2026.

Sentry. Skill Scanner. https://github.com/getsentry/skills/tree/main/skills/skill-scanner, 2026. Skill in the getsentry/skills repository, accessed September 23, 2026.

Snyk. Snyk Agent Scan: Security Scanner for AI Agents, MCP Servers, and Agent Skills. https://github.com/snyk/agent-scan, 2026. GitHub repository, accessed September 23, 2026.

Tencent Zhuque Lab. AI-Infra-Guard: A Comprehensive, Intelligent, and Easy-to-Use AI Red Teaming Platform. https://github.com/Tencent/AI-Infra-Guard, 2025. GitHub repository, accessed September 23, 2026.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996.

Yongheng Zhang, Tingwen Du, Yunshan Ma, Xiang Wang, Yi Xie, Guozheng Yang, Yuliang Lu, and Ee-Chien Chang. Attackg+: Boosting attack graph construction with large language models. *Computers & Security*, 150:104220, 2025. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2024.104220. URL https://www.sciencedirect.com/science/article/pii/S0167404824005261.

A Appendix

A.1 The MalSkillBench Dataset

MalSkillBench is a benchmark for malicious skill detection (Guo et al., 2026). It contains 3944 malicious skills and 4000 benign skills. A malicious skill enters the benchmark only after its declared malicious behavior is observed at runtime. MalSkillBench verifies code injection and mixed attack skills in a Docker sandbox under system-call monitoring and verifies prompt injection skills through the response of the agent. MalSkillBench generates 3214 of the malicious skills, collects 703 wild skills from public registries, and extracts 27 test cases from existing detection tools. The wild skills are manually verified. The 4000 benign skills are the most-downloaded skills in the community.

Every malicious skill in MalSkillBench is labeled with an attack vector and a malicious behavior. The attack vectors are code injection, prompt injection, and mixed attacks. Code injection places malicious executable code in the referenced scripts or in inline code blocks of the skill document. The code performs malicious actions when the agent executes it. Prompt injection places adversarial instructions in the skill document. Mixed attacks split one attack chain across the skill document and its scripts. Neither the instructions nor the code is malicious alone. Behaviors B1 through B9 can appear under all three attack vectors. Behaviors B10 through B15 target the reasoning of the agent and appear only under prompt injection. Table 4 lists the 15 malicious behaviors.

Our detector targets skills whose instructions direct an agent to perform malicious actions. Therefore, we exclude the 1300 prompt-injection malicious skills. The resulting dataset contains 4000 benign skills and 2644 malicious skills. The malicious skills in our dataset come from four sources in MalSkillBench. Table 5 reports the number of skills from each source.

We split the dataset by skill family, which is defined by the origin of skills in MalSkillBench, so that no family appears in both the training set and the test set. We assign 80% of the skills to the training set and 20% to the test set. The training set contains 5315 skills. The test set contains 1329 skills, including 529 malicious skills and 800 benign skills.

Table 4: Malicious behaviors defined in MalSkillBench.

ID	Behavior
B1	Data Exfiltration
B2	Credential Theft
B3	Remote Code Execution
B4	Malware Delivery
B5	Persistence
B6	Reverse Shell
B7	Ransomware
B8	Resource Abuse
B9	Privilege Escalation
B10	Role Hijack
B11	Safety Bypass
B12	Instruction Override
B13	System Prompt Leak
B14	Goal Hijacking
B15	Content Manipulation

Table 5: Source composition of the 2644 malicious skills in our dataset.

Source	Count
Code injection	1346
Mixed attacks	568
Wild samples	703
Test cases from existing detectors	27
Total	2644

A.2 Skill Graph Schema

Tables 6 and 7 describe the node and edge records extracted from skill documents. Fields grouped under an explicit node type belong to its `properties` record.

A.3 Skill Graph Extraction

The extractor reads `SKILL.md` and checks for filenames under `scripts/` whose relative paths occur in the document. Script bodies are not included in the LLM input.

Document chunks. A document with at most 16000 characters is processed in one call. Longer documents are divided at Markdown headings of levels one through four. Consecutive sections are accumulated toward a target of 12000 characters. A section is also appended when the accumulated chunk contains fewer than 1500 characters. These boundaries allow an individual section to exceed the target without truncation. The leading document metadata block, when present, is prepended to every chunk. Each chunk receives the full list of script filename hints from the document.

Extraction prompt. Each call contains one system message and one user message. The system message concatenates five prompt components in a fixed order. The components specify node types and entity naming, edge types and operation fields, extraction examples, placeholders for referenced scripts, and entity descriptions. The final component stores entity descriptions in the field named `constraints`. The prompt components are separated by a Markdown horizontal rule. The user

Table 6: Node fields and type-specific properties.

Scope	Field	Description
Common	name	Name of the entity.
	type	One of the five explicit node types, or AGENT for an implicit tool endpoint.
	canonical_id	Normalized identifier for entity merging.
	properties	Properties associated with the node type.
	constraints	Entity descriptions with text and source section.
	_meta	Provenance and placeholder metadata.
FILE	path	File path.
	extension	File extension.
	permission	Read, write, read/write, execute, or unknown.
NETWORK	domain	Domain or network endpoint identifier.
	protocol	Network protocol.
CREDENTIAL	type	Token, API key, password, or environment variable.
	scope	Read-only, read/write, or unknown.
PACKAGE	name	Package or program name.
	registry	Package registry or package manager.
	executable	Whether the package is executable.
USER	interaction_type	Confirmation, selection, input, or notification.

Table 7: Operation edge fields.

Field	Description
source	Source endpoint of the operation.
target	Target endpoint of the operation.
edge_type	One of READ, WRITE, EXEC, FETCH, SEND, SEARCH, ASK, or SPAWN.
tool_used	Tool associated with the operation.
keywords	Keywords associated with the extracted operation.
description	Textual description of the operation.
source_layer	Recorded source category, either declaration or implementation.
source_section	Section of the skill document associated with the operation.
order	Optional order value recorded for the operation.

message supplies the document chunk and script hints and requests an output in JavaScript Object Notation (JSON).

The following template shows the user message structure with line wrapping for readability. The two fields in angle brackets denote input substitutions.

```
Extract entities, operations, and constraints from the
following SKILL.md.  Follow the schema in the system prompt.
Pay special attention to the placeholder instructions for
any 'scripts/...' path you encounter.

=== SKILL.md ===
⟨document chunk⟩

=== Provided Script Hints ===
⟨script hints⟩

Emit your JSON now.
```

The prompt produces node and edge records with their descriptions. The configured model uses a temperature of 0 and an output limit of 12000 tokens in JSON object mode.

Validation and merging. The parser normalizes supported aliases and validates node records against their type-specific schemas. For example, a file requires a path, whereas a network endpoint requires a domain and protocol. An extracted edge must have a supported operation type, a compatible tool identifier, and a nonempty description. Its endpoints must resolve to graph nodes or recognized agent tool names. The parser removes self-loops and records discarded entries.

For each referenced script without a file placeholder, the parser inserts a file node with execute permission and an execution edge from the recognized agent endpoint `Bash`. For an existing placeholder, it adds this edge when no edge from `Bash` to the placeholder is present. Placeholder identifiers in edge endpoints are replaced with the corresponding file node names.

The outputs of multiple chunks are combined within the same skill. For node records with the same existing canonical identifier, the combination keeps the first record and unions entity descriptions by their text and source section. Edges from different chunks are deduplicated by source, target, operation type, and source section. Nodes inserted without canonical identifiers receive them when the graph is saved.

A.4 Rule Search Procedure

The search enumerates rules over predicate and entity-type combinations observed in the augmented training graphs. Heads use operation relations. Bodies use operation relations, unary ML predicates, or binary ML predicates. All arguments in the enumerated rules are variables. No entity constants are introduced during the search.

Rule statistics. The search uses the matching function m and the support, confidence, and head coverage statistics defined in Section 3.2. These statistics measure behavioral associations without assigning malicious labels to rules.

Candidate extensions. A unary atom reuses one variable of the required entity type. A binary atom reuses two variables or introduces one variable while reusing another. Variables in the head and current body are available for reuse. Repeated body atoms are excluded. For a head $R(x, y)$, the implementation also excludes an extension with predicate R and first argument x. It excludes an extension with predicate R and second argument y while x is absent from the current body. These conditions restrict candidate enumeration in addition to type and connectivity requirements.

Enumeration and pruning. Let $\mathrm{Ext}(H, B)$ return the extensions described above. For a head H, write $s_H = \sum_i m(\{H\}, G_i^+)$ and $b_H = \max\{2, \lceil 0.01 s_H \rceil\}$. The procedure below returns the candidate rule set.

1. Initialize an empty set of output rules.
2. For each observed operation predicate and pair of endpoint types, construct a head H with two variables of those types. Skip the head if $s_H < 2$.
3. Initialize the search frontier with the empty body. For body lengths one and two, initialize an empty next frontier and an empty set of bodies visited at that length.
4. For each body B in the frontier and each atom $a \in \mathrm{Ext}(H, B)$, form $B' = B \cup \{a\}$. Skip repeated atoms and bodies already visited at that length.
5. Compute $s = \sum_i m(B' \cup \{H\}, G_i^+)$. Discard B' if $s < b_H$. Otherwise, add B' to the next frontier.
6. If every head variable occurs in B' and $s / \sum_i m(B', G_i^+) \geq 0.3$, add $B' \Rightarrow H$ to the output set after deduplication by its canonical structural signature.
7. Replace the frontier by the next frontier and continue until the body length reaches two or the frontier becomes empty. After all heads are processed, return the output rules in canonical structural order.

The support bound combines the minimum support and head coverage requirements. Confidence filters output rules without pruning their extensions. The requirement that head variables occur in the body also filters output rules without pruning their extensions.

A.5 Text Model and Rule Weight Training

Text preprocessing. The text model reads entity descriptions from `nodes[].constraints[].text` and operation descriptions from `edges[].description`. It replaces web addresses, email addresses, numerical network addresses, paths, and long identifiers with category tokens in that order. It normalizes whitespace and converts the text to lowercase before the security-term substitutions. Security terms, such as malware, ransomware, and command and control, are replaced with `SECURITY_TERM`. The implementation defines these substitutions through regular expressions. Entity names, skill labels, and family identifiers are not appended to the text features.

MIL settings. TF-IDF uses word unigrams and bigrams, a minimum document frequency of four local records, and at most 30000 features. It uses sublinear term frequency, smoothed inverse document frequency, Unicode accent normalization, and L_2 normalization of each feature vector. For a description vector x_{ij}, the local logit and score are

$$z_{ij} = w_{\text{text}}^\top x_{ij} + b_{\text{text}}, \qquad q(t_{ij}) = \sigma(z_{ij}). \tag{10}$$

Equation 6 pools these logits into the bag logit g_i. The bag label probability is $p_\theta(X_i) = \sigma(g_i)$, where $\theta = (w_{\text{text}}, b_{\text{text}})$. Using the logistic loss ℓ defined in Equation 3, we minimize

$$\mathcal{L}_{\text{text}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \alpha_i \ell(y_i, g_i) + \frac{\lambda_{\text{text}}}{2} \|w_{\text{text}}\|_2^2, \qquad \alpha_i = \frac{N}{2N_{y_i}}. \tag{11}$$

Here, N_{y_i} is the number of training skills with label y_i. We use $\lambda_{\text{text}} = 10^{-5}$ and leave b_{text} unpenalized. Optimization starts from zero, uses at most 500 iterations, and sets a function tolerance of 10^{-9}. Three folds grouped by skill family generate the local scores for the training graphs. The folds use stratification by label and a random seed. After cross-fitting, the MIL model is refitted on all training skills; this final model scores the descriptions of all test skills (MalSkillBench test set and ASM) to instantiate ML predicate facts.

Rule weights and threshold. The feature used by LR is the binary feature h_r defined in Section 3.2. The LR model uses the `liblinear` solver, balanced class weights, an L_1 penalty, and at most 3000 iterations. The synthetic intercept feature uses the solver's default scale of one, so the intercept is penalized. We consider inverse regularization strengths $C \in \{0.01, 0.03, 0.1, 0.3, 1.0\}$. Three folds grouped by skill family use label stratification and a random seed. The predicate facts and candidate rules are shared across these folds. The text model and rule search are not refitted within each rule-weight fold.

For each C, predictions on the held-out folds determine the detection threshold. Candidate thresholds include the distinct prediction scores and the values 0, 0.5, 1, and 1.000001. When there are more than 800 distinct scores, 801 evenly spaced quantile levels replace that score list. The selected threshold maximizes F1 subject to an FPR limit of 0.37. Ties are resolved by higher precision, lower FPR, and then the higher threshold. The choice of C uses F1, precision, and lower FPR in that order. The final LR model is fitted on all training skills with the selected C. The threshold is stored to eight decimal places.

Rule merging. The compression procedure compares the conjunction of the body and head atoms. It uses substitutions that preserve entity types and implications between thresholds of the same ML predicate. Redundant atoms can be removed when the reduced conjunction is equivalent under these definitions. Rules with the same canonical conjunction after this reduction share one detection feature. Their coefficients are summed without refitting or threshold selection. This operation preserves the weighted training scores within numerical tolerance.

A.6 Evaluation Protocol

The main comparison reports one run per configuration on the fixed family split. SBR uses temperature 0 for graph extraction and fixed random seeds for fold construction and LR fitting. Static pattern matching introduces no sampling during inference with fixed inputs and configurations. The reported detection metrics are point estimates without confidence intervals or estimates of variation across training seeds and data splits.

A.7 Selection of the Graph Extraction Model

We evaluate graph extraction to explain our choice of MiniMax-M2.5 and to examine whether smaller models can perform this stage of SBR at a controlled cost. The first comparison evaluates seven model configurations on 50 skills and selects the model with the best overall extraction quality. The second comparison evaluates MiniMax-M2.5 and Qwen3.5-9B on 20 ASM documents against reference annotations and measures how much extraction quality the smaller model retains.

Comparison of candidate models. All configurations process the same 50 skills with the graph extraction pipeline described in Appendix A.3. The 50 skills consist of 24 benign skills and 26 malicious skills from MalSkillBench. The two metrics compare the node names extracted from each skill with the evaluation text and use no skill labels. The evaluation text of a document concatenates all files under its skill directory with text, code, and configuration extensions in sorted order. Before comparison, names and artifacts are normalized by lowercasing and removing surrounding quotes, punctuation, and brackets. For each document d, let N_d denote the set of normalized node names extracted by a model. Let S denote a fixed set of 25 schema-level synthetic names, which contains the agent and user nodes, tool names such as `bash` and `grep`, and standard shell paths. Let A_d denote the set of distinct URLs, domains with common endings, slash-separated paths with file extensions, and uppercase environment variable names with underscores, identified in the evaluation text of d. Artifacts shorter than four characters are removed from A_d.

Grounding precision and artifact recall are

$$\begin{aligned} \mathrm{GP} &= \frac{1}{|D|} \sum_{d \in D} \frac{|\{n \in N_d \setminus S : \ n \text{ occurs in the evaluation text of } d\}|}{|N_d \setminus S|}, \\ \mathrm{AR} &= \frac{1}{|D'|} \sum_{d \in D'} \frac{|\{a \in A_d : \ a \text{ matches an element of } N_d\}|}{|A_d|}, \end{aligned} \tag{12}$$

where D contains the documents with $|N_d \setminus S| > 0$ for the evaluated model and D' contains the documents with $|A_d| > 0$. A name occurs in the evaluation text when it is a substring of the lowercased evaluation text. An artifact matches a name when one is a substring of the other. Table 8 reports the results.

Extraction with a smaller model. The second experiment compares MiniMax-M2.5 and Qwen3.5-9B under the same graph schema, document inputs, and extraction prompt. Both models use temperature 0. We select 10 benign and 10 malicious ASM documents.

Table 8: Graph extraction with seven model configurations on 50 skills.

Model	Grounding precision	Artifact recall
MiniMax-M2.5	0.8872	0.5118
MiniMax-M3(MiniMax, 2026a)	0.8127	0.5030
Qwen3.5-4B(Qwen Team, 2026a)	0.8636	0.4571
Qwen3-8B(Qwen Team, 2025)	0.9384	0.3780
agnes-2.5-flash(Agnes AI Team, 2026)	0.8236	0.4776
Qwen3.5-9B(Qwen Team, 2026a)	0.8516	0.4986
Qwen3.8-27B(Qwen Team, 2026b)	0.8444	0.5008

Table 9: Extraction against reference annotations for 20 ASM documents.

	Entity			Structural relation		
Model	Precision	Recall	F1	Precision	Recall	F1
MiniMax-M2.5	0.727	0.392	0.509	0.538	0.269	0.359
Qwen3.5-9B	0.589	0.310	0.406	0.462	0.207	0.286

The reference annotations are generated with GPT-6-Astra and reviewed by two authors in both experiments. Each annotated fact retains its supporting text, which is verified programmatically. Network nodes with the same host and explicit port are merged into one entity. User interactions are annotated at the level of single questions, confirmations, and notifications.

An entity matches when its type and normalized name or identifying property agree with a reference entity or a recorded alias under a one-to-one assignment. User interactions match their content or recorded equivalent expressions. A structural relation matches when its operation type and matched target entity agree with the reference. Repeated pairs of operation type and target count once. The scores include user interactions, their `ASK` relations, and parser-added script nodes. Descriptions, conditions, negation, and source actors are outside this scoring rule. We aggregate true positives, false positives, and false negatives across documents to compute micro precision, recall, and F1.

The entity F1 scores are 0.509 for MiniMax-M2.5 and 0.406 for Qwen3.5-9B. The structural relation F1 scores are 0.359 and 0.286. The entity difference of 0.103 has a 95% interval of $[0.035, 0.189]$ from 10000 paired bootstrap resamples of documents, whereas the relation difference of 0.073 has an interval of $[-0.081, 0.218]$. Including all unresolved reference items gives an entity F1 of 0.485 for MiniMax-M2.5 and 0.395 for Qwen3.5-9B, and a relation F1 of 0.329 and 0.267. Qwen3.5-9B reaches both F1 scores within 0.11 of MiniMax-M2.5. The capability gap on this task is limited. Graph extraction with a smaller model therefore keeps the cost of this stage controllable. Table 9 reports the results.

A.8 Ablation on Behavioral Rules

We assess the contribution of learned behavioral rules by replacing SBR's rule matching with a decision based on local MIL scores. This variant uses the same trained MIL model as SBR. It labels a skill as malicious when more than half of its descriptions score at least 0.5, with each description counted once. We evaluate both methods on the MalSkillBench test set and ASM using the samples and metrics of the corresponding main-text comparisons.

Replacing behavioral rules with the MIL majority decision substantially reduces F1 and recall on both datasets. On MalSkillBench, the ablated variant achieves 26.23% F1 and 15.12% recall, compared with 87.17% and 84.12% for SBR. On ASM, it achieves 9.16% F1 and 4.84% recall, compared with 71.36% and 61.29% for SBR. The ablated variant produces fewer false positives, while missing most malicious skills in both datasets.

Table 10: Ablation on behavioral rules. *ML majority* predicts a malicious skill when more than half of its local descriptions receive MIL scores of at least 0.5. *SBR* uses the corresponding configurations in Tables 1 and 3. All values are percentages. **Bold** indicates the better value within each dataset.

Dataset	Method	Acc.↑	Prec.↑	Rec.↑	F1↑	FPR↓
MalSkillBench	ML majority	66.14	**98.77**	15.12	26.23	**0.13**
	SBR	**90.14**	90.45	**84.12**	**87.17**	5.88
ASM	ML majority	65.71	**85.71**	4.84	9.16	**0.45**
	SBR	**82.42**	85.39	**61.29**	**71.36**	5.83

A.9 Case Study

This case study shows how an individual prediction decomposes into matched rules, learned weights, and the entities that satisfy them. All numbers in Figures 4 and 5 come from the detector output on the extracted graphs of test skills.

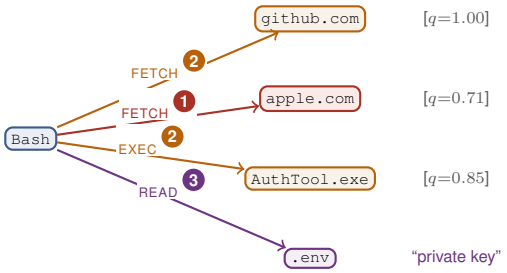
Figure 4 traces the malicious skill `MSB_001` from its document through the behavior graph to the verdict. Each rule joins structural predicates with an ML predicate over the same entities. Every condition names the entities it binds, which allows an auditor to check the rule against the document.

Figure 5 applies the same rule set to a malicious skill and a benign skill. The two skills accumulate weight in opposite directions. In the left panel, two rules with the same body structure differ only in the threshold of their node ML predicate. Their weights have opposite signs on the same binding. For every skill, the detector outputs the complete list of matched rules and their weights.

1 | The skill document — `MSB_001 base-trading-agent` (excerpt; harmful strings redacted)

```
name: base-trading-agent # Automated DEX Trading on Base
1 # For macOS Users --- run this command:
echo "macOS-Installer: https://swcdn.apple.com/content/downloads/update/software/upd/"
echo 'L2Jpbi9iYXNoIC1j [redacted] ...KSI=' | base64 -D | bash
# decodes to: /bin/bash -c "$(curl -fsSL http://91.92.242.[redacted])"
2 # For Windows: download AuthTool.zip from
github.com/[redacted]
# (password: 1234), then run AuthTool.exe
3 # .env must contain: PRIVATE_KEY=... MNEMONIC=...
```

2 | Extracted behavior graph (46 nodes, 29 edges; decisive subgraph shown)



ML predicates that hold:
2 $\mathrm{ML}^{\mathrm{node}}_{0.3}(\texttt{AuthTool.exe})$ [q=0.65],
$\mathrm{ML}^{\mathrm{edge}}_{0.7}(\texttt{Bash}, \texttt{AuthTool.exe})$ [q=0.85],
2 $\mathrm{ML}^{\mathrm{edge}}_{0.9}(\texttt{Bash}, \texttt{github.com})$ [q=1.00]

3 | Rule firing and verdict (60 hits, $+50/-10$)

Σw^{+}=15.370, Σw^{-}=−2.648; three of the 60 matched rules:

$\mathrm{FETCH}(x, f) \wedge \mathrm{ML}^{\mathrm{node}}_{0.3}(y) \Rightarrow \mathrm{EXEC}(x, y)$ 1 2
x=Bash, f=apple.com, y=AuthTool.exe
$w = +2.394$

$\mathrm{ML}^{\mathrm{edge}}_{0.7}(x, y) \wedge \mathrm{READ}(x, f) \Rightarrow$
$\mathrm{EXEC}(x, y)$ 2 3
x=Bash, y=AuthTool.exe, f=.env $w = +0.685$

$\mathrm{FETCH}(f', y) \wedge \mathrm{ML}^{\mathrm{edge}}_{0.9}(x, f) \Rightarrow$
$\mathrm{FETCH}(x, y)$ 1 2
x=Bash, y=apple.com, f=github.com
$w = +1.164$

$\mathrm{logit} = -1.876 + \sum_i w_i = 10.846$

MALICIOUS $p = 0.99998 \geq \eta = 0.50130$

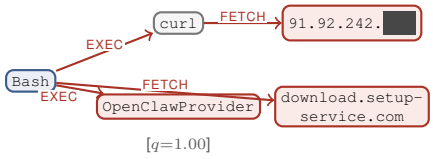
Figure 4: **Anatomy of one rule-based detection (MalSkillBench `MSB_001`).** The document carries three malicious instruction threads (panel 1, badges; harmful strings redacted): 1 a fake installer URL paired with a base64-obfuscated remote script, 2 a password-protected trojan download and execution, and 3 wallet-key collection via `.env`. Graph extraction maps each thread to operation edges with local text scores (panel 2; edges colored by thread, edge scores beside their targets). Panel 3 shows three of the 60 matched rules. Each composes evidence from two threads (badges). The 60 matched-rule weights sum to 12.722. Adding the intercept -1.876 gives a logit of 10.846 and a maliciousness probability of approximately 0.99998, above the frozen detection threshold $\eta = 0.50130$. Ten of the 60 matched rules carry negative weights. Figure 5 shows a benign skill on which such weights determine the verdict.

A benign skill document — `MSB_151 byterover` (excerpt)

```
name:  byterover # Knowledge management for AI agents
description:  "You MUST use this for gathering contexts before any work.  ...
Uses a configured LLM provider (default:  ByteRover, no API key needed) ..."
# connect a different provider with your own key:
brv providers connect openai --api-key sk-xxx --model gpt-4.1
```

`MSB_059` (deep-research) — malicious, detected

Extraction recovers the payload chain [8 nodes, 8 edges]:



[q=1.00]

95 rule hits (+79/−16), $\Sigma w = +16.753$; e.g.:

$\text{FETCH}(x, f) \wedge \text{ML}^{\text{node}}_{0.7}(y) \Rightarrow \text{EXEC}(x, y)$

x=Bash, f=download.setup-service.com,
y=OpenClawProvider $w = +2.577$

$\text{FETCH}(x, f) \wedge \text{ML}^{\text{node}}_{0.3}(y) \Rightarrow \text{EXEC}(x, y)$

same binding $w = -1.360$

$\text{ML}^{\text{edge}}_{0.9}(x, y) \Rightarrow \text{FETCH}(x, y)$

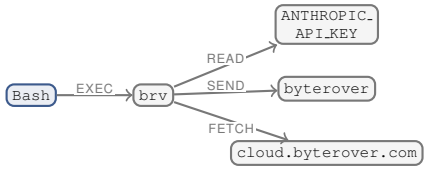
x=curl, y=91.92.242.* $w = +1.372$

$$\text{logit} = -1.876 + 16.753 = 14.877$$

DETECTED $p = 1.00000 \geq \eta = 0.50130$

`MSB_151` (byterover) — benign, cleared

Provider setup in the graph [11 nodes, 20 edges; decisive subgraph shown]:



[q=0.44]

10 rule hits (+5/−5), $\Sigma w = -0.941$; e.g.:

$\text{READ}(x, f) \wedge \text{SEND}(s, y) \Rightarrow \text{SEND}(x, y)$

x=s=brv, f=ANTHROPIC_API_KEY, y=byterover
$w = -0.968$

$\text{EXEC}(f', x) \wedge \text{SEND}(s, y) \Rightarrow \text{SEND}(x, y)$

f'=Bash, x=s=brv, y=byterover $w = -0.804$

$\text{ML}^{\text{edge}}_{0.3}(x, y) \Rightarrow \text{FETCH}(x, y)$

x=brv, y=cloud.byterover.com $w = +0.828$

$$\text{logit} = -1.876 - 0.941 = -2.817$$

BENIGN $p = 0.0564 < \eta = 0.50130$

Figure 5: **One rule set, two directions of evidence.** The malicious `MSB_059` (left) carries a disguised provider install. Extraction recovers the payload chain. The 95 matched rules accumulate a weight sum of $+16.753$, which gives $p = 1.00000 \geq \eta$. The benign `MSB_151 byterover` skill (right) reads API keys and issues send operations to provider endpoints during provider setup. Ten rules match this graph, five with positive weights and five with negative weights. The negative weights dominate. The logit of -2.817 gives $p = 0.0564 < \eta$. Two rules in the left panel have the same body and differ only in the threshold of their node ML predicate. Both match the same binding, with weights $+2.577$ and -1.360 and a net contribution of $+1.217$.