StarCore: The Model that Dreams the Cybersecurity World

StarCore Team

Metaordo & Tsinghua University

(For the complete list of authors, please refer to the Contribution section)

Abstract

We present StarCore, a next-generation world model designed to transition the paradigm of world models to the cybersecurity world. Building upon the capabilities of learning multimedia data in the system world, including code, commands, traffic, and agent traces, StarCore adopts the technique of system state chain learning to significantly improve the performance of understanding and analyzing the dynamic data streams behind diverse network systems. To advance the model in the dynamic system world, we implement a new state-chain learning framework for system data that models the system world's states with diverse benign and malicious behaviors and introduces three state-chain supervised fine-tuning tasks for system world understanding. Furthermore, we propose a novel RL framework based on GRPO algorithms with state-chain rewards that further improve the quality of understanding system state chains and reasoning for security analysis. Through these innovations, StarCore achieves state-of-the-art performance on state chain completion, prediction, and security analysis tasks. Most critically, StarCore demonstrates unprecedented capability in understanding dynamic real-world system information streams, surpassing traditional foundation models that rely on simple QA reasoning for cybersecurity. Code, models, and more information are available at https://github.com/metaordo/StarCore.

1 Introduction

With the rapid development of Internet and artificial intelligence, cyberspace has evolved from a traditional information-exchange environment into a complex dynamic system world composed of software codes, operation systems, network protocols, business applications, and autonomous agents. In parallel, cyber attacks have become increasingly stealthy, intelligent, and sophisticated. Attack behaviors are no longer confined to a single vulnerability exploit or a fixed attack pattern. Instead, they have evolved into multi-stage campaigns that continuously unfold around the running state of the target system.

Mainstream cyber-defense technologies rely on human-written rules, signature matching, and machine-learning classifiers that analyze a single type of data at a single point in time. For instance, Intrusion Detection Systems (IDS) [2][8][24] recognize anomalous patterns in network traffic with predefined rule bases or learned models. Security Information and Event Management (SIEM) [18] platforms correlate logs, alerts, and events across hosts. Provenance-based threat-hunting systems reconstruct attack campaigns by correlating suspicious information flows and kernel audit records [16].. Although effective within their own scopes, these approaches treat each data modality as an isolated island which produces static entity-relationship snapshots rather than a model of how system states actually transition over time.

To enhance the intelligence and automation of network security analysis, large language models [12] [22] [6] [23] [3] have begun to be introduced to reshape security operations. Microsoft Security Copilot [15] combines an LLM with security data sources and a tool-calling orchestration layer for alert analysis and threat explanation. Cisco [5] continually pre-trains a Llama-3.1 [22]

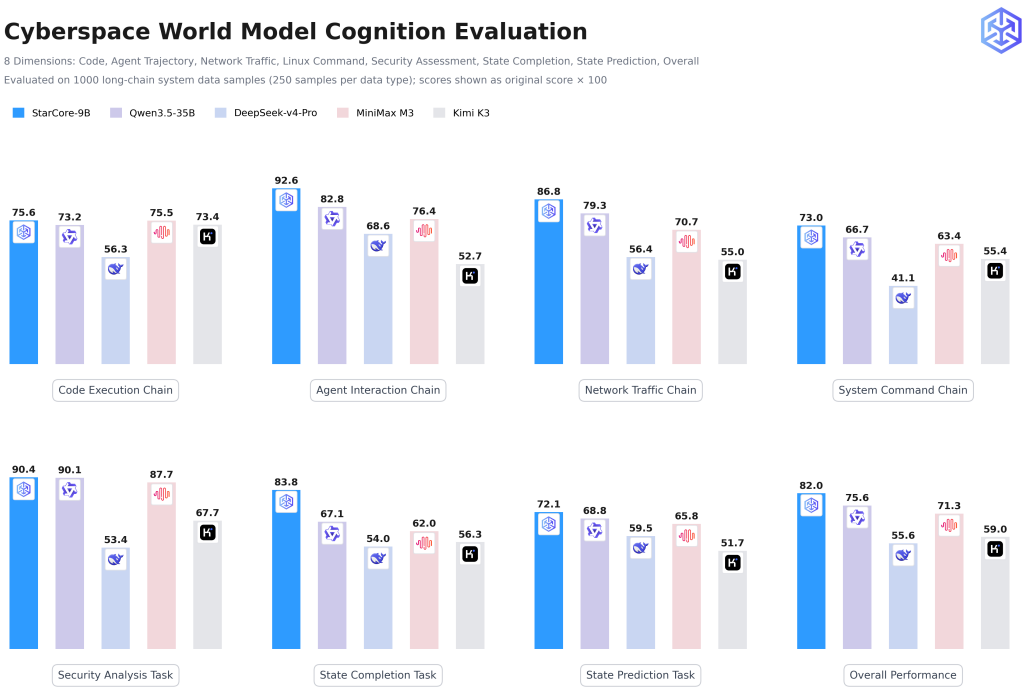


Figure 1: Results of StarCore-9B, Qwen3.5-35B, DeepSeek-v4-Pro, Minimax M3, and Kimi K3 on 8 cyberspace world cognition benchmarks: Code Execution Chain, Agent Interaction Chain, Network Traffic Chain, System Command Chain, Security Analysis Task, State Completion Task, State Prediction Task, and Overall Performance Benchmark.

backbone on CVE, CWE, ATT&CK, and threat-intelligence corpora to obtain a security-knowledge reasoning model. Despite their strong capabilities in knowledge comprehension and textual reasoning, these models are trained on static text, and their knowledge representation is dominated by semantic relations among documents. They can answer what a vulnerability affects or summarize what an attack log contains, but they cannot perceive how an attack action alters subsequent system states, or how one running state transitions into another. In short, they know security without understanding the system world.

Design. We present StarCore, a cybersecurity world model built upon system state chain learning. StarCore takes the system data streams within a network system world as the minimal world set to be understood. The heterogeneous data produced during system operation, including execution code, system commands, network traffic, and agent interaction trajectories, are treated as world-state descriptions. Within each modality, consecutive states are chained along the system execution order into execution state chains, forming large scale of long-distance directed chains that structurally represents the system running world and captures the attack propagation within the system state evolution. Through a new state-chain learning mechnism, StarCore models the system world's states with diverse benign and malicious behaviors and introduces novel state-chain learning tasks with SFT and RL techniques for system world understanding. It demonstrates a concrete path from knowledge-driven security question answering toward world-model-driven security understanding and deduction, enabling attack-chain tracing, attack-evolution prediction, and dynamic risk analysis for proactive defense.

Methods. StarCore instantiates a capability progression of from seeing the system world to understanding the system world. First, the model undergoes security-knowledge continued post-training to internalize vulnerability, attack, and threat-intelligence knowledge. Second, we perform state-chain supervised fine-tuning (SFT) to learn state-transition regularities through three state-chain learning tasks, including state chain completion, state chain prediction, and state chain security analysis, combining with explicit reasoning traces for system state understanding. Third, we align the model

with a state-chain reinforcement learning method. Instead of optimizing textual answer matching as in conventional RLHF, we take the correctness of state-chain reasoning as the reward basis, design a multi-dimensional State Chain Reward (SCR) that scores output format, thinking quality, and answer correctness to optimize the policy with Group Relative Policy Optimization (GRPO) [20]. In summary, the leap from security knowledge to world understanding in StarCore is driven by the following technical contributions:

- First, we propose a state-chain modeling method for the network system world. StarCore defines a multimodal unified state space over execution code, traffic flows, system commands, and agent traces, These system execution state chains are constructed to transform static and isolated event records into coherent, and temporally ordered behavioral stories of the system world.
- Second, we present a state-chain-supervised world-understanding learning paradigm. Rather than learning question-answer mappings over security knowledge, StarCore is supervised directly with system state chains through completion, prediction, and security-analysis tasks, with large scales of long-distance state reasoning for world understanding, thereby internalizing the evolutionary regularities, causal mechanisms, and cross-modal associations of network system states.
- Third, we adopt a state-chain reinforcement learning alignment method for deep-thinking security models. StarCore utilizes a multi-dimensional State Chain Reward (SCR) method to evaluate system state-chain understanding capabilities for different modalities and employs the GRPO algorithm to explore diverse reasoning paths and the possibility of diverse states in running systems, substantially improving generalization to unseen attack variants and complex dynamic environments.

With these advancements, StarCore overthrows traditional foundation model paradigms that rely on simple QA reasoning for cybersecurity and stands as a world model that captures dynamic real-world system information streams for security-task reasoning. We release StarCore to the community to further advance the frontier of fundation models for network security.

2 System State-Chain Modeling

The first pillar of StarCore is a state-chain modeling method that turns raw, heterogeneous system data into state chains of the system world. We proceed in two steps: constructing a unified system state space, and organizing states into system state chains.

2.1 A Unified State Space for the System World

System World. The system world is the minimal world set of different system data streams, such as execution codes, network traffic, and system commands, together with the interaction trajectories of agents running on top of them. It comprises the entities, behaviors, and environmental changes involved in the operation of a network system.

World Model. The world model [10] [14] [11] aims to develop a foundational model capable of comprehending the operational states and interrelationships within the realm of network systems, while predicting future attack behaviors. The model will construct and interpret the dynamic environment formed by aggregating system data and leverage this understanding to analyze and forecast future attacks.

System State Chain. To achieve such capability, we define the data-chain scope of the system world to cover four key modalities that span the complete behavioral spectrum from low-level binary execution to high-level agent interaction, including the chain of code execution, network communications, system command execution, and agent behaviors. To model the state transition process of the dynamic network system world, we first formulate the run-time data of four modalities as system states and then represent the complete state transition process as a state chain, thereby capturing massive normal and malicious system behaviors throughout the system world.

Given the system data that represents the current state of the system o_t at the time step t, which comes from the last state of system data o_{t-1} with the benign or malicious system action a_{t-1}, where $a_t \in \mathcal{A}$ denotes the system action space that running behind the observed system data, the world model

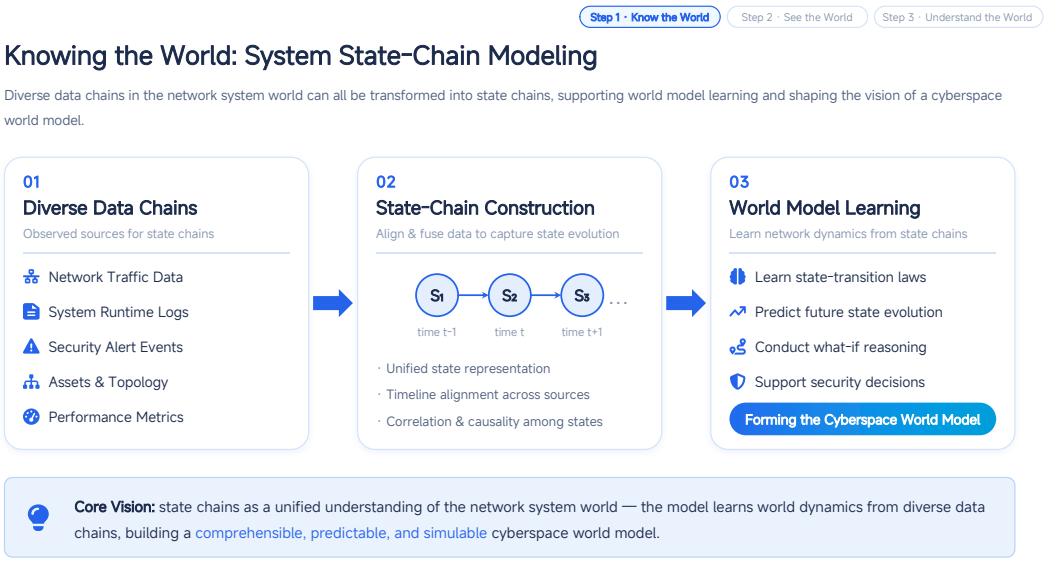


Figure 2: **Modeling the world:** StarCore models the data transition knowledge observed in the running systems as the system state chains to build the world view of the cyberspace.

$G_{WM}(o_t, a_t)$ should model the complete system transition trajectory $\tau = \{o_0, a_0, o_1, a_1, ..., o_T\}$. For a time step t, we define the system data chain as $c = \{o_0, o_1, ..., o_T\}$ to help the model capture the system running actions $\alpha = \{a_0, a_1, ..., a_{T-1}\}$ with different benign or attack intentions.

2.2 Uni-modal System State Chain

Within a single modality, behaviors are continuous, temporally ordered, and governed by evolution regularities. We therefore concatenate consecutive states of the same modality along the time axis to form uni-modal system state chains, each being a complete behavioral sketch of an entity in that modality. In this paper, we use four types of chains shown as follows:

- **Code Execution Chain.** Tracing the instruction-execution trajectory of a program or script run, the code states of sequentially executed basic blocks or code lines are concatenated along the control flow, revealing the dynamic path from entry to exit.
- **Network Traffic Chain.** Around one network session or a group of related flows, packet or flow states are concatenated in chronological order, characterizing the full communication sequence from handshake and negotiation through data transfer to teardown.
- **System Command Chain.** Over the lifecycle of a user session or service process, the log states of executed commands or audit events are concatenated by timestamp, reflecting the operator's action sequence and evolving intent.
- **Agent Interaction Chain.** Within one agent task session, the states of dialog turns or tool-call steps are concatenated, describing the agent's continuous decisions, observations, and environment interactions.

Figure 3 shows the details of building the different types of system state chain. We use the system state chain to model the dynamic state transition occurred in a real-world network system. Through this mapping, originally isolated code fragments, packets, log entries, and dialog turns become a set of synchronized state snapshots living in a common space, which for the first time gives the upper-layer model a cross-modal, panoramic perception of the running system.

2.3 Multi-modal System State Chain

Once attacks are executed cross multi-model chains, complex system interactions usually cannot be captured with statistic QA and limited windows of traditional LLMs. Therefore, the world model aims to also cover a wide range of system states with multi-modal system interaction chains. For

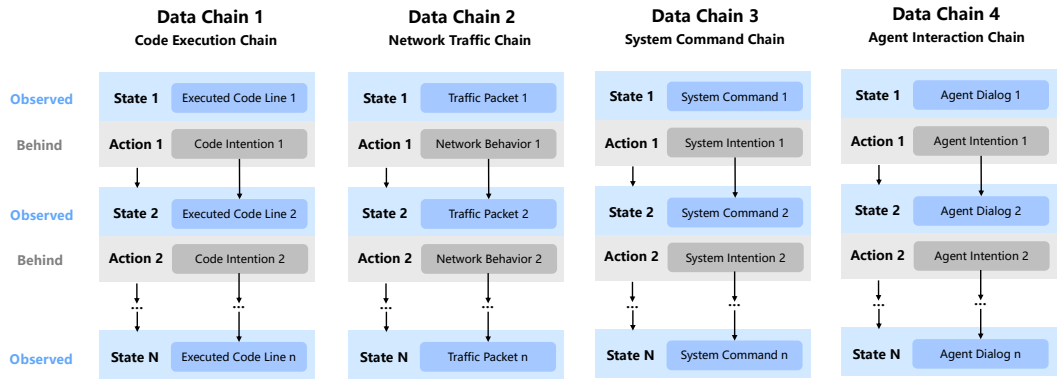


Figure 3: Illustration of the system state-chain construction among four types of data chains.

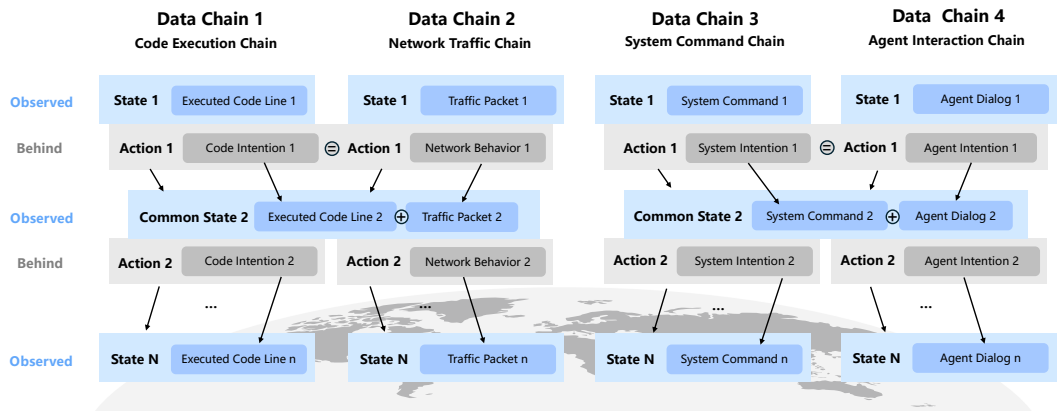


Figure 4: Illustration of the interaction graph build by multi-modal system state chains.

example, a complex attack usually simultaneously triggers multiple system state chains, such as remote shell attacks may triggers both system command chains and network traffic chains, which produce stealthy command traffic when creating remote shell and executing data exfiltration.

World Prototype: Interaction Graph with System State Chains. Modeling with large-scale multi-modal system state chains, the world model will face an extra network space of system state chains to form the interaction graph in the cyberspace. Figure 4 depicts the illustration of the interaction graph. Once a malicious attack or benign behavior involves multiple system data chain, the equivalent actions of different data chains will produce the intersection points as the common states, which connect two or more system state chains to build the interaction graph of the cyberspace world.

Through this mapping, originally isolated code fragments, packets, log entries, and dialog turns become a set of synchronized state snapshots living in a common space, which for the first time gives the upper-layer model a cross-modal, panoramic perception of the running system.

3 State-Chain Supervision Fine-Tuning

The supervision fine-tuning phase of StarCore aims to transform the traditional LLMs into a highly capable world model with cyberspace data-chain reasoning abilities. As illustrated in Figure 5, we propose a state-chain supervision fine-tuning (SFT) pipeline to use system state chains rather than knowledge question-answer pairs as the supervision target, lifting the model from perceiving system states to understanding them. By leveraging the unique training paradigm and three state-chain learning tasks, StarCore effectively achieves system data-chain completion, future event prediction, and detailed security analysis, bridging the reasoning gap of existing models in long-chain reasoning under the realistic cyberspace world.

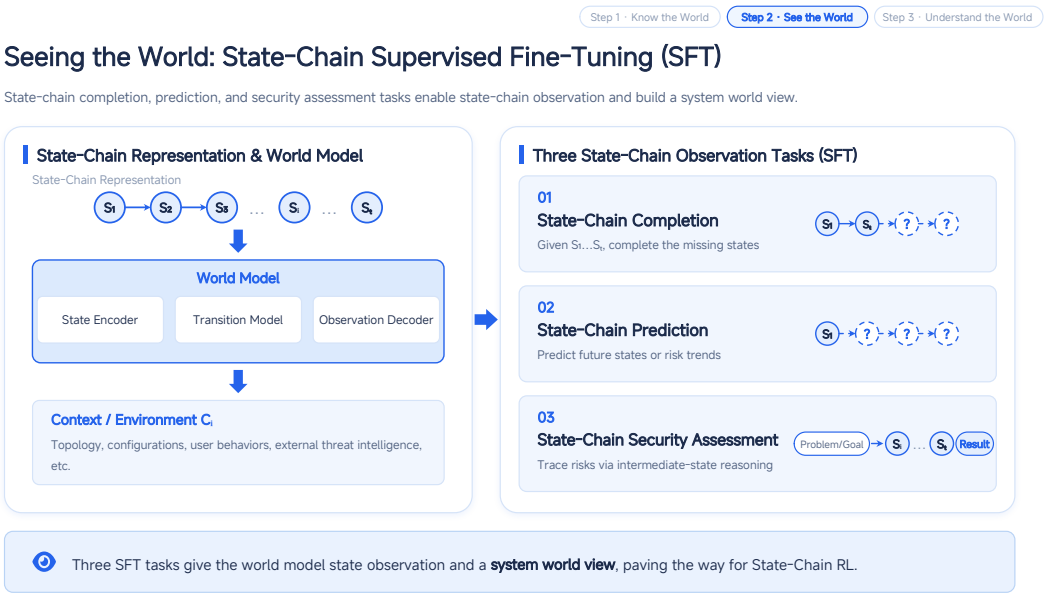


Figure 5: **Seeing the world:** StarCore receives the system state chains by seeing the context of the chains with three specific state-chain supervised fine-tuning tasks.

3.1 Training Paradigm

Compared with traditional LLMs, StarCore significantly expands the scale of observation and reasoning for the system state chains during the SFT stage. The state-chain SFT data structure of StarCore covers five major categories:

- **Instruction**: The predefined system instruction that describe the task to conduct or the role to play in the cybersecurity area.
- **Question**: The user input that include the question based on the cyberspace system data.
- **States**: The states are the core system state chains should be observed by the world model, which could include code, traffic data, system running log, agent trajectory, and other data that can be modeled as system state chains.
- **Action Reasoning**: The reasoning of the actions is the evidence the world model find based on the observation of the states, which describe the possible actions behind the state sequence and explain why the state transition appeared and what the action intents are.
- **Answer**: The answer gives the final decision of the world model to solve the complete analysis task queried by users.

The SFT corpus of StarCore covers three major categories: open-sourced datasets, self-collected datasets, and AI-generated datasets. StarCore utilizes large-scale security-related QA and CVE/CWE database to build the foundation knowledge of the world model. To model various state chains of a running system, we widely collect billions of codes, traffic data, command logs, and agent trajectory from public-available repositories on the Internet, and produce real-world system running data from controlled cloud servers. To enable state-of-the-art reasoning ability, StarCore uses Qwen3.5-9B as the base model, and learns from reasoning results from well-known LLMs including GLM-5.2 [21], MiniMax M3 [17], DeepSeek-V4-Pro [7], and Kimi K3 [13] as the reference of action reasoning, which gives the chance to combine frontier LLM knowledge for cybersecurity reasoning.

3.2 State-Chain Learning Tasks

During the state-chain learning stage, we teach the world model to observe the system through state chains. Three supervised fine-tuning tasks are then constructed through state-chain completion, state-chain prediction, and state-chain security assessment, giving the world model genuine state-observation ability and establishing its system world view.

StarCore employs three state-chain learning tasks to realize state-chain supervised fine-tuning:

- **State Chain Completion**: The state chain completion task requires the world model determining the missed states based on the context, building the understanding of the complete transition.
- **State Chain Prediction**: The state chain prediction task requires the world model to predict the next state based on historical chains, forming abilities of guessing future action in early stages.
- **State Chain Security Assessment**: The state chain security assessment task requires the world model to analyse the possible abnormal behaviors or malicious intentions behind the data chains.

3.3 State Chain Completion

The state chain completion task aims to enable the world model to recover missing system states by understanding the contextual dependencies and evolution patterns within an observed state chain. In real-world network systems, system observations are often incomplete due to packet loss, insufficient logging, partial data collection, or limited visibility into system execution. These missing observations may obscure critical behavioral transitions and break the continuity of the system state chain. We use $S = \{s_0, ..., s_T\}$ to model the system state chains. C represents the context and environment of the system. Therefore, the world model should learn to infer unobserved states from their preceding and succeeding states, rather than simply memorizing individual observations.

$$\arg\max_{S_{\mathcal{M}}} P_\theta(S_{\mathcal{M}} \mid \widetilde{S}, C), \tag{1}$$

where $S_{\mathcal{M}} = s_t \mid t \in \mathcal{M}$ denotes the ground-truth missing states and θ represents the parameterized world model. We use R to model the reasoning of the world model. $\mathcal{D}$ is the SFT datasets of the state chain completion task.

$$-\mathbb{E}(S, C, R) \sim \mathcal{D}\left[\log P_\theta(S_{\mathcal{M}}, R_{\mathcal{M}} \mid \widetilde{S}, C)\right]. \tag{2}$$

Through this task, the world model learns to recover missing observations by exploiting long-range temporal dependencies, contextual correlations, and behavioral continuity across system states. More importantly, it develops the ability to infer the latent system activities behind incomplete observations, establishing a foundation for understanding the underlying dynamics of the cyberspace world.

3.4 State Chain Prediction

The state chain prediction task aims to equip the world model with the ability to anticipate future system states based on historical observations. Unlike state chain completion, which reconstructs missing states within an existing trajectory, state chain prediction focuses on forecasting states that have not yet been observed. In dynamic network environments, benign operations and malicious activities continuously evolve over time, and their subsequent behaviors are often reflected in changes to code execution, network communications, system execution, and agent interactions. K is the future steps after state S_t. Learning these evolution patterns enables the world model to anticipate potential system behaviors and identify emerging security risks before they fully materialize.

$$\prod_{j=1}^{K} P_\theta(s_{t+j} \mid S_{j-1}, C). \tag{3}$$

This formulation encourages the model to learn the transition regularities of system states, where each predicted state is conditioned on both historical observations and previously predicted states.

$$-\mathbb{E}(S, C, R) \sim \mathcal{D}\left[\log P_\theta(S_{t+K}, R_{t+K} \mid S_0, C)\right]. \tag{4}$$

By learning to predict future states and their underlying behavioral intentions, the world model captures the temporal evolution patterns of running systems. This capability allows StarCore to reason beyond currently observed system states and anticipate subsequent system behaviors, providing a basis for early-stage attack discovery and proactive cybersecurity analysis.

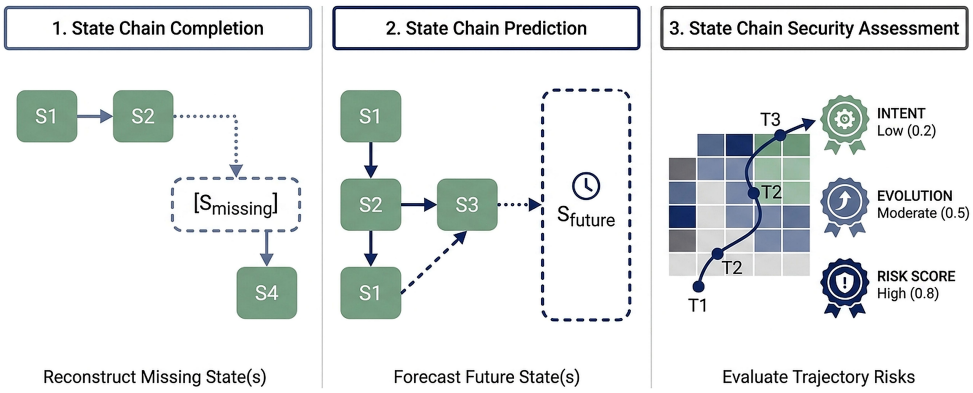


Figure 6: Schematic of the three state-chain supervised fine-tuning (SFT) tasks: (1) State Chain Completion for inferring unobserved intermediate steps; (2) State Chain Prediction for projecting future attack paths; and (3) State Chain Security Assessment for behavioral intent scoring.

3.5 State Chain Security Assessment

The state chain security assessment task aims to enable the world model to understand the security implications of system state evolution and identify potential threats through long-range behavioral reasoning. Unlike conventional security analysis that classifies isolated events or individual observations, this task requires the model to analyze the complete state chain and investigate the relationships among intermediate states. In real-world cyberspace environments, malicious activities may be disguised as legitimate operations at individual time steps, while their underlying attack intentions become evident only through the accumulated behavioral evidence and the interactions among multiple system states. Therefore, effective security assessment requires the world model to trace behavioral transitions, identify abnormal activities, and explain the security implications of the observed system evolution.

$$P_\theta(R \mid S, C, Q) \cdot P_\theta(Y \mid R, S, C). \tag{5}$$

Here, Q is the instruction of a complex security tasks. R represents the intermediate reasoning process for identifying relevant states, tracing their behavioral dependencies, and inferring possible system actions and intentions. Y denotes the final assessment, which summarizes the identified security behaviors, potential attack intentions, and corresponding risks based on the observed evidence.

$$-\mathbb{E}(S, C, R) \sim \mathcal{D}\left[\log P_\theta(R, Y \mid S, C, Q)\right]. \tag{6}$$

By jointly learning intermediate action reasoning and final security assessments, the world model develops the ability to connect individual observations into coherent behavioral explanations. This task enables StarCore to move beyond static event classification toward state-chain-based security understanding, supporting attack behavior tracing, malicious intention identification, and evidence-based risk assessment in dynamic cyberspace environments. Figure 6 shows the illustration of all the SFT tasks.

4 State-Chain Reinforcement Learning

The state-chain supervised fine-tuning stage enables StarCore to observe system states, reconstruct incomplete state chains, predict future system behaviors, and perform security analysis. However, supervised learning primarily optimizes the likelihood of predefined reference trajectories, which may limit the model's ability to explore diverse reasoning paths and handle complex, previously unseen system behaviors. To further enhance the world model's understanding of the dynamic cyberspace, we introduce a state-chain reinforcement learning (RL) framework based on Group Relative Policy Optimization (GRPO). Rather than optimizing the textual similarity of generated answers alone, our framework directly evaluates the quality of system state-chain understanding and behavioral reasoning through a multi-dimensional State Chain Reward (SCR).

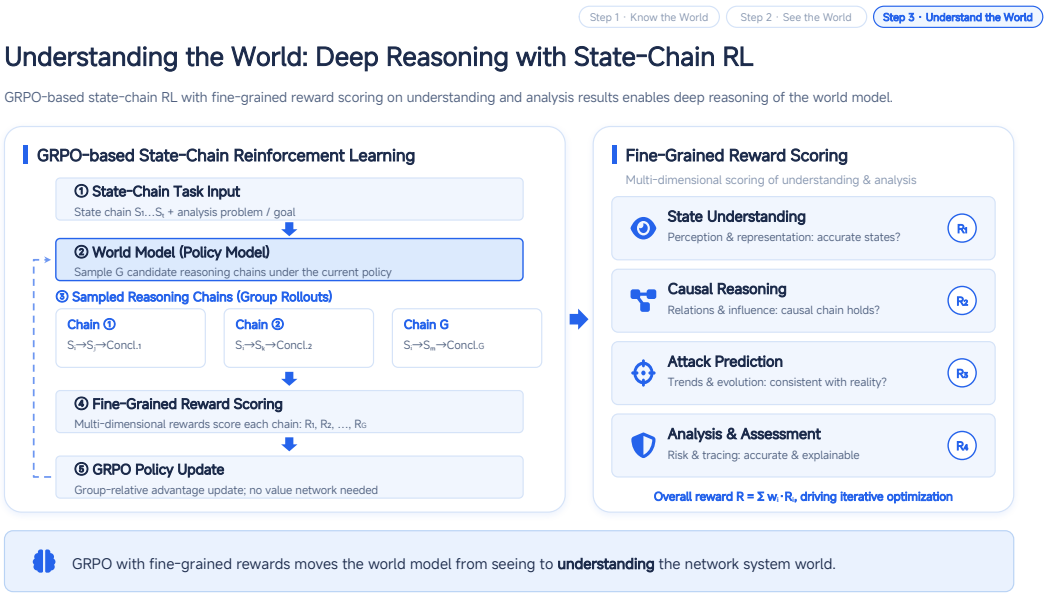


Figure 7: **Understanding the world:** StarCore employs GRPO to realize deep reasoning with state-chain reinforcement learning, which enables the robust reasoning ability for long-chain analysis.

Specifically, we formulate state-chain reasoning as a reward-driven generation process, where the world model explores multiple candidate reasoning trajectories for each system state-chain input. We design SCR to evaluate the generated trajectories from complementary dimensions, including state understanding, causal reasoning, attack prediction, and security assessment, while considering output format validity. Based on these fine-grained rewards, we employ GRPO to optimize the policy model through group-relative advantage estimation and clipped policy updates, without introducing an additional value model. This process progressively aligns the world model with the underlying evolution regularities of the cyberspace, improving its ability to conduct long-chain reasoning, understand complex system behaviors, and identify potential security threats.

4.1 State-Chain Reinforcement Learning Formulation

We formulate state-chain reinforcement learning as a reward-driven conditional generation problem. Given a system state chain, its corresponding environmental context, and a task instruction, the world model is required to generate a reasoning trajectory and a final response that accurately characterize the observed system behaviors. Unlike conventional security-oriented reinforcement learning that primarily evaluates the correctness of a final classification or textual answer, our formulation considers the intermediate reasoning process as an essential component of the optimization objective.

Formally, we define a state-chain RL input as

$$x = (\mathcal{I}, Q, S, C, \tau), \tag{7}$$

where $\mathcal{I}$ denotes the system instruction, Q represents the task-specific question or analysis objective, $S = \{s_1, s_2, \ldots, s_T\}$ denotes the observed system state chain, C represents the corresponding system context and environmental information, and $\tau \in \mathcal{T}$ indicates the state-chain learning task. Here, $\mathcal{T} = \{\text{comp}, \text{pred}, \text{assess}\}$ represents state chain completion, prediction, and security assessment, respectively.

The world model is parameterized by a policy π_θ, which generates a response trajectory $o = (R, Y)$ conditioned on the input x. Specifically, R represents the intermediate reasoning trace that identifies relevant state evidence, infers potential system actions, and explains the behavioral relationships among system states. Y denotes the final task output, including reconstructed states, predicted future states, or security assessment results, depending on the task type. The conditional generation

probability is formulated as

$$\pi_\theta(o \mid x) = \prod_{t=1}^{|o|} \pi_\theta(o_t \mid x, o_{<t}), \tag{8}$$

where o_t denotes the t-th generated token, $o_{<t}$ represents all preceding generated tokens, and $|o|$ is the response length.

The objective of state-chain RL is to maximize the expected State Chain Reward over the distribution of state-chain inputs:

$$\max_\theta \ \mathcal{J}(\theta) = \mathbb{E}_{x \sim \mathcal{P}_{\mathrm{sc}},\, o \sim \pi_\theta(\cdot|x)} \left[R_{\mathrm{SCR}}(x, o) \right], \tag{9}$$

where $\mathcal{P}_{\mathrm{sc}}$ denotes the distribution of state-chain RL inputs and $R_{\mathrm{SCR}}(x, o)$ evaluates the quality of the generated reasoning trajectory and its corresponding output.

This formulation encourages the world model to discover high-quality reasoning trajectories rather than merely reproduce fixed reference responses. By evaluating both intermediate behavioral reasoning and final task outcomes, state-chain RL provides a direct optimization signal for the model's understanding of system dynamics and security implications.

4.2 State Chain Reward Design

The quality of a cybersecurity world model cannot be sufficiently characterized by the correctness of its final textual answers. In particular, a model may produce a seemingly correct security conclusion while overlooking critical intermediate states, misinterpreting behavioral dependencies, or generating an unsupported attack explanation. To address this limitation, we propose a multi-dimensional State Chain Reward (SCR) that evaluates the model's understanding and reasoning capabilities throughout the state-chain analysis process.

As illustrated in Figure 7, SCR considers five complementary reward components, covering output format validity, state understanding, causal reasoning, attack prediction, and security assessment. These components jointly evaluate the model's ability to perceive system states, understand their evolution, anticipate potential behaviors, and derive reliable security conclusions.

Output Format Reward. The output format reward evaluates whether the generated response follows the required structural and serialization constraints. Since state-chain tasks involve heterogeneous system data and structured reasoning results, well-formatted outputs are essential for reliable interpretation and downstream processing. We define the format reward as

$$r_{\mathrm{fmt}}(x, o) = \mathcal{V}_{\mathrm{fmt}}(o), \tag{10}$$

where $\mathcal{V}_{\mathrm{fmt}}(\cdot)$ is a format verifier that evaluates the structural validity and completeness of the generated response.

State Understanding Reward. The state understanding reward measures whether the model correctly identifies and interprets the key information contained in the observed system state chain. It evaluates the model's perception of relevant entities, system behaviors, state attributes, and temporal relationships. In particular, the model should distinguish normal system activities from suspicious observations without overlooking critical evidence. We denote this reward as

$$r_{\mathrm{state}}(x, o) = \mathcal{V}_{\mathrm{state}}(S), \tag{11}$$

where $\mathcal{V}_{\mathrm{state}}(\cdot)$ evaluates the consistency between the generated state understanding and the available system-state evidence.

Causal Reasoning Reward. The causal reasoning reward evaluates whether the model correctly explains the behavioral dependencies and potential actions underlying system state transitions. Instead of simply identifying suspicious states, the model should establish a coherent behavioral trajectory by connecting preceding and succeeding observations and explaining how potential system actions lead to the observed changes. We define

$$r_{\mathrm{causal}}(x, o) = \mathcal{V}_{\mathrm{causal}}(S, R), \tag{12}$$

where $\mathcal{V}_{\mathrm{causal}}(\cdot)$ evaluates the logical consistency of the inferred behavioral relationships, the supporting state evidence, and the corresponding action reasoning. This reward encourages the model

to develop coherent explanations of system evolution rather than generate isolated or unsupported behavioral descriptions.

Attack Prediction Reward. The attack prediction reward measures the model's ability to anticipate future system behaviors and identify potential attack developments from historical state chains. It evaluates whether the predicted states and behavioral trends are consistent with the actual system evolution, particularly for emerging or multi-stage malicious activities. We formulate this reward as

$$r_{\text{pred}}(x, o) = \mathcal{V}_{\text{pred}}(S_{\text{future}}, R, Y), \tag{13}$$

where S_{future} denotes the reference future system states when available, and $\mathcal{V}_{\text{pred}}(\cdot)$ evaluates the consistency of the generated predictions and their underlying behavioral reasoning with the reference evolution.

Security Assessment Reward. The security assessment reward evaluates the correctness and evidential support of the final security analysis. It considers whether the model accurately identifies abnormal behaviors, infers potential malicious intentions, and derives security conclusions consistent with the observed state-chain evidence. Importantly, the model should avoid unsupported attack attribution and distinguish legitimate system activities from actual malicious behaviors. We define

$$r_{\text{assess}}(x, o) = \mathcal{V}_{\text{assess}}(S, R, Y), \tag{14}$$

where $\mathcal{V}_{\text{assess}}(\cdot)$ evaluates the correctness, completeness, and evidence consistency of the generated security assessment.

The overall SCR is computed as a weighted aggregation of these reward components:

$$\begin{aligned} R_{\text{SCR}}(x, o) = w_{\text{fmt}} r_{\text{fmt}} + w_{\text{state}} r_{\text{state}} \\ + w_{\text{causal}} r_{\text{causal}} + w_{\text{pred}} r_{\text{pred}} \\ + w_{\text{assess}} r_{\text{assess}}, \end{aligned} \tag{15}$$

where each reward component is normalized to a comparable range, and w_{fmt}, w_{state}, w_{causal}, w_{pred}, and w_{assess} are non-negative weights controlling their respective contributions. The weights can be configured according to the relative importance of each evaluation dimension.

Since different state-chain tasks emphasize different capabilities, the reward components are activated according to the task type τ. For example, completion emphasizes the correctness of reconstructed states, prediction emphasizes the consistency of future-state forecasts, and security assessment emphasizes behavioral reasoning and security conclusions. Accordingly, the general reward formulation can be expressed as

$$R^{(\tau)}_{\text{SCR}}(x, o) = \sum_{k \in \mathcal{K}} m_{\tau,k} w_k r_k(x, o), \tag{16}$$

where $\mathcal{K}$ denotes the set of reward dimensions, and $m_{\tau,k} \in \{0, 1\}$ indicates whether the k-th reward component is applicable to task τ. The weights of the active components are normalized such that

$$\sum_{k \in \mathcal{K}} m_{\tau,k} w_k = 1. \tag{17}$$

Through this multi-dimensional reward design, SCR evaluates not only whether the model produces a correct answer, but also whether it understands the observed system states, establishes reasonable behavioral relationships, and provides evidence-supported security reasoning. This enables the reinforcement learning process to optimize the underlying world-understanding capabilities rather than merely improve surface-level answer generation.

4.3 GRPO-based Policy Optimization

Based on the proposed SCR, we employ Group Relative Policy Optimization (GRPO) to optimize the state-chain reasoning policy. GRPO is particularly suitable for our framework because it enables the model to explore multiple reasoning trajectories for the same system state-chain input and directly compare their relative quality using group-level rewards, without requiring an additional value network.

Specifically, given a state-chain input x, the current policy $\pi_{\theta_{\text{old}}}$ independently samples a group of G candidate response trajectories:

$$\mathcal{O}(x) = \{o_1, o_2, \ldots, o_G\}, \qquad o_i \sim \pi_{\theta_{\text{old}}}(\cdot \mid x), \tag{18}$$

where θ_{old} denotes the parameters of the policy used to generate the current rollout group, and G is the number of sampled candidates. Each candidate consists of an intermediate reasoning trace and a final task response. We then evaluate each candidate using SCR to obtain its scalar reward:

$$r_i = R_{\text{SCR}}(x, o_i), \qquad i = 1, \ldots, G. \tag{19}$$

To estimate the relative quality of each candidate, GRPO normalizes the rewards within the sampled group. Specifically, the group mean and standard deviation are calculated as

$$\mu_r = \frac{1}{G} \sum_{i=1}^{G} r_i, \tag{20}$$

$$\sigma_r = \sqrt{\frac{1}{G} \sum_{i=1}^{G} (r_i - \mu_r)^2}. \tag{21}$$

The group-relative advantage of the i-th response is subsequently defined as

$$\hat{A}_i = \frac{r_i - \mu_r}{\sigma_r + \epsilon}, \tag{22}$$

where ϵ is a small constant for numerical stability. A positive advantage indicates that a candidate performs better than the group average, whereas a negative advantage indicates relatively inferior performance. This relative comparison encourages the policy to increase the probability of higher-quality reasoning trajectories while reducing the probability of inferior ones.

To optimize the policy while preventing excessively large updates, we adopt the clipped surrogate objective used in GRPO. For the t-th token of the i-th sampled response, the probability ratio between the updated policy and the rollout policy is defined as

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid x, o_{i,<t})}. \tag{23}$$

The clipped policy optimization objective is formulated as

$$\begin{aligned} \mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{x, \mathcal{O}(x)} \Bigg[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \min \Big(& \rho_{i,t}(\theta) \hat{A}_i, \\ & \text{clip}\left(\rho_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right) \hat{A}_i \Big) \Bigg], \end{aligned} \tag{24}$$

where ε is the clipping parameter that constrains the magnitude of policy updates, and $\text{clip}(\cdot)$ restricts the probability ratio to the interval $[1-\varepsilon, 1+\varepsilon]$. The sequence-level advantage $\hat{A}_i$ is shared across all generated tokens of the corresponding response, allowing the policy to optimize the overall quality of each reasoning trajectory.

Furthermore, to preserve the security knowledge and state-chain understanding acquired during the preceding training stages, we introduce a Kullback–Leibler (KL) divergence constraint between the updated policy and a reference policy π_{ref}, which is initialized from the state-chain SFT model and remains frozen during RL optimization. The final GRPO objective is given by

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathcal{J}_{\text{clip}}(\theta) - \beta \mathbb{E}_{x, o \sim \pi_\theta} \left[D_{\text{KL}} \left(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x) \right) \right], \tag{25}$$

where β controls the strength of the KL regularization. This constraint prevents the optimized policy from deviating excessively from the reference model, thereby retaining its acquired cybersecurity knowledge and state-chain understanding capabilities while improving its reasoning quality.

The GRPO optimization proceeds iteratively. At each iteration, the policy generates multiple candidate reasoning trajectories for each state-chain input, evaluates them using SCR, computes group-relative advantages, and updates the policy by maximizing the clipped objective with KL regularization. Unlike conventional actor–critic methods, this process does not require an additional value model to estimate token-level or state-level values, reducing the additional parameters and training overhead.

Overall, the proposed state-chain reinforcement learning framework combines multi-dimensional behavioral evaluation with group-relative policy optimization. By explicitly rewarding accurate state understanding, coherent behavioral reasoning, reliable attack prediction, and evidence-based security assessment, StarCore progressively improves its ability to understand the evolution of complex system behaviors. This enables the model to move beyond merely observing and reproducing system state chains toward deeply reasoning about the dynamic cyberspace and underlying security implications.

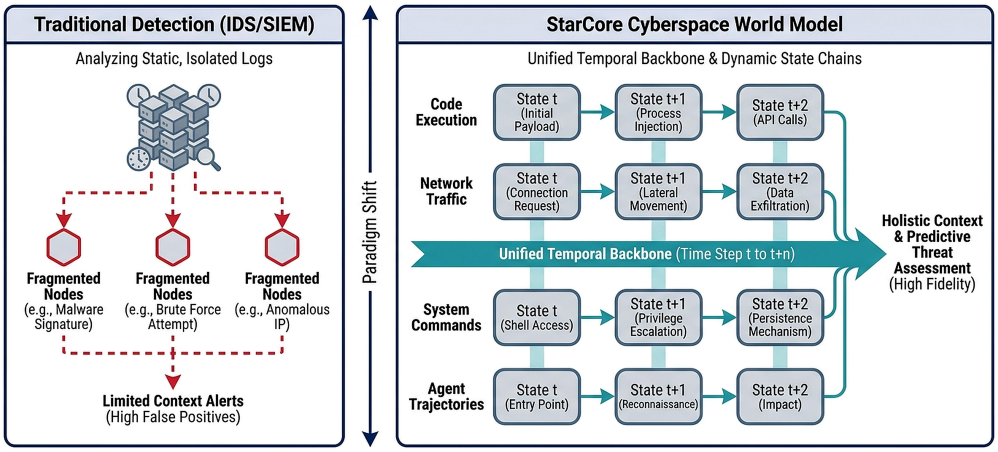


Figure 8: Comparison between traditional cybersecurity threat detection and the world model paradigm. Unlike static tools analyzing isolated logs, StarCore continuously perceives dynamic state chains spanning code execution, network traffic, system commands, and agent trajectories over unified time horizons.

5 Evaluation

5.1 Experimental Setup

Hardware Environment. We develop and evaluate StarCore on two GPU platforms, NVIDIA B300 and A800. These platforms provide the computational resources for model development, training, and inference evaluation. We conduct experiments across both platforms to support the development and testing of the proposed cybersecurity world model under different hardware configurations.

Dataset Collection. To construct a comprehensive corpus for cybersecurity world modeling, we collect publicly available data from multiple open-source platforms and research resources, including Hugging Face, GitHub, and ModelScope, as well as open-source repositories and datasets released through scientific research in the cybersecurity community. Rather than relying on a single data source or a specific data format, our collection covers diverse artifacts generated by real-world computing systems and security-related activities.

Data Modalities. The collected corpus spans four major modalities: (1) *source code*, including program implementations and code-related artifacts; (2) *network traffic*, including packet- and flow-level data; (3) *operating system logs*, including system events and execution records; and (4) *agent trajectories*, including interaction histories and action-related records produced by intelligent agents. These modalities capture complementary aspects of system behavior, ranging from program logic and network communication to runtime events and agent interactions. By integrating heterogeneous data sources, StarCore is designed to learn system states and their temporal and causal dependencies across different cybersecurity scenarios. Figure 8 shows the world model's data chain paradigm compared to traditional models.

Data Preparation. Before training and evaluation, the collected data are organized into structured state-chain samples following the data format described in Section 3.1. Each sample is associated with task instructions and corresponding questions, together with the available system states, action reasoning, and expected answers. This unified representation enables heterogeneous cybersecurity artifacts to be incorporated into the state-chain learning tasks, including state-chain completion, state-chain prediction, and state-chain security assessment.

5.2 Overall Performance Evaluation

We first evaluate the answer accuracy of StarCore by comparing it with a diverse set of state-of-the-art large language models, including Qwen, DeepSeek, MiniMax, Kimi, and GLM. The evaluation covers four system data modalities, namely code, agent trajectories, network traffic, and Linux commands, as well as three core system-world understanding tasks: security analysis, state completion, and

Table 1: Answer accuracy comparison of different LLMs across system data modalities and system-world understanding tasks. Overall accuracy is calculated as the arithmetic mean of the seven evaluation dimensions. The best and second-best results in each column are highlighted in bold and underlined, respectively.

Model	System Data Types				System-World Understanding Tasks			Overall
	Code	Agent	Traffic	Command	Security	Completion	Prediction	
Qwen3.5-9B	0.7496	0.7525	0.7724	0.6508	0.8845	0.6426	0.6596	0.7328
Qwen3.5-35B	0.7325	0.8279	0.7931	0.6669	0.9012	0.6709	0.6878	0.7561
DeepSeek-v4-Flash	0.5771	0.6345	0.0945	0.2345	0.4776	0.3796	0.2962	0.3847
DeepSeek-v4-Pro	0.5632	0.6860	0.5636	0.4112	0.5343	0.5398	0.5946	0.5563
MiniMax M3	0.7550	0.7636	0.7071	0.6337	0.8771	0.6201	0.6584	0.7134
Kimi K3	0.7344	0.5272	0.5496	0.5539	0.6765	0.5632	0.5170	0.5902
GLM-5.2	0.7175	0.7980	0.6436	0.5165	0.7731	0.6258	0.6069	0.6688
GLM-5.3	0.6329	0.5696	0.6084	0.5569	0.8269	0.4652	0.4820	0.5919
Qwen3-235B-A22B	0.7931	0.5049	0.6604	0.3860	0.6000	0.6714	0.5387	0.6090
Qwen3.8-Flash-Next	0.6950	0.7472	0.5957	0.6440	0.8176	0.6051	0.5893	0.6705
GLM-5.3-Flash	0.6980	0.6273	0.6851	0.6018	0.8358	0.5318	0.5901	0.6531
StarCore-9B	0.7564	0.9260	0.8678	0.7297	0.9040	0.8380	0.7207	0.8201
StarCore-35B	0.8929	0.9104	0.9339	0.8117	0.8716	0.9122	0.8781	0.8872
StarCore-180B	**0.9306**	**0.9603**	**0.9672**	**0.8371**	**0.9429**	**0.9301**	**0.8983**	**0.9238**
StarCore-320B	0.8904	0.9470	0.9100	0.7738	0.9313	0.9237	0.7851	0.8802

state prediction. For each evaluation dimension, answer accuracy measures the proportion of model-generated answers that are correct. Table 1 reports the answer accuracy of different models across these seven dimensions, together with their overall accuracy.

Overall Answer Accuracy. As shown in Table 1, StarCore achieves consistently high answer accuracy across different model scales. In particular, StarCore-180B obtains the highest overall accuracy of 0.9238, outperforming the strongest baseline, Qwen3.5-35B (0.7561), by 16.77%. This result demonstrates the effectiveness of state-chain learning in improving the correctness of model-generated answers when interpreting heterogeneous system behaviors and reasoning about their security implications. Moreover, the accuracy improvements span all seven evaluation dimensions, indicating that the benefits of StarCore are not confined to a particular data modality or task.

Answer Accuracy across System Data Modalities. We further examine answer accuracy across the four system data modalities. StarCore-180B achieves accuracies of 0.9306, 0.9603, 0.9672, and 0.8371 on code, agent trajectories, network traffic, and Linux commands, respectively, outperforming all baseline models in each dimension. Compared with the strongest baseline in each category, these results represent improvements of 13.75, 13.24, 17.41, and 17.02%, respectively. In particular, the high accuracy on network traffic and Linux commands highlights the model's ability to correctly interpret operational system behaviors beyond conventional code understanding. These results demonstrate that StarCore can produce reliable answers across heterogeneous system data modalities, rather than being restricted to a specific type of system data.

Answer Accuracy on System-World Understanding Tasks. Beyond modality-specific understanding, we evaluate whether models can correctly answer questions involving higher-level system-world reasoning through security analysis, state completion, and state prediction. StarCore-180B achieves accuracies of 0.9429, 0.9301, and 0.8983 on these three tasks, respectively, consistently outperforming all baseline models. In particular, it improves answer accuracy on state completion and prediction by 25.87 and 21.05%, respectively, compared with the strongest baseline in each task. These substantial improvements highlight the benefits of learning system-state dependencies and temporal evolution. Rather than merely interpreting individual system artifacts, StarCore can leverage contextual information to correctly reconstruct missing states and predict subsequent system behaviors. Meanwhile, its 0.9429 accuracy on security analysis demonstrates its strong capability to interpret system behaviors and produce correct security assessments.

Answer Accuracy across Model Scales. We also evaluate StarCore at different model scales, including 9B, 35B, 180B, and 320B parameters. Even at the 9B scale, StarCore-9B achieves an

Table 2: Comparison of reasoning correctness across different LLMs on the BigModelGym benchmark. Higher values indicate more accurate reasoning rationales. The best and second-best results in each row are highlighted in bold and underlined, respectively.

	StarCore-9B	**MiniMax M3**	**GLM-5.2**	**Qwen3.5-35B**	**Qwen3.5-9B**	**DeepSeek v4-Flash**	**DeepSeek v4-Pro**	**Kimi K3**
Overall	**52.8**	<u>47.7</u>	44.0	43.9	42.7	40.2	39.8	39.6
System-World Tasks								
Security Tasks	**57.0**	48.0	<u>49.8</u>	47.9	49.1	35.9	44.4	35.7
Completion Tasks	**60.2**	<u>55.7</u>	44.2	50.9	47.7	38.4	43.0	44.6
Prediction Tasks	41.1	38.4	38.0	32.5	31.1	**46.3**	32.0	36.6
System Data Types								
Code Data	**76.4**	<u>71.0</u>	62.5	63.7	61.3	55.4	56.5	55.5
Agent Trajectory	**42.2**	34.3	35.9	39.2	37.9	38.4	33.1	<u>38.7</u>
Traffic Data	**66.6**	<u>57.5</u>	55.3	52.2	52.4	28.6	52.6	45.4
System Command	25.9	26.8	22.3	21.2	18.2	**38.4**	16.8	20.3

overall accuracy of 0.8201, outperforming Qwen3.5-9B (0.7328) by 8.73%. At the 35B scale, StarCore-35B further achieves an overall accuracy of 0.8872, substantially outperforming Qwen3.5-35B (0.7561). The 180B variant achieves the best overall accuracy of 0.9238, indicating that the proposed state-chain learning framework can deliver substantial improvements across different model scales. Although the 320B variant remains competitive with an overall accuracy of 0.8802, its accuracy is lower than that of the 180B variant. This observation suggests that increasing model size alone does not necessarily lead to higher answer accuracy. Overall, these results demonstrate that the improvements of StarCore are not solely attributable to parameter scaling, but also reflect the benefits of explicitly learning system-state dependencies and dynamics.

5.3 Reasoning Capability Evaluation

In addition to evaluating answer accuracy, we further investigate whether StarCore can produce correct and logically grounded reasoning processes when understanding complex system behaviors. To this end, we conduct an independent evaluation of reasoning correctness using the StarCore benchmark on BigModelGym [4]. Unlike answer evaluation, which measures the correctness of final answers, this evaluation independently assesses whether the reasoning rationales generated by models are correct. As described by the benchmark, each evaluation dimension contains 250 samples or corresponding sample combinations, with 1,000 samples in each leaderboard. The reasoning evaluation covers four system data types (code, agent trajectories, network traffic, and system commands) and three system-world understanding tasks (security analysis, state completion, and state prediction). The overall score is calculated using the benchmark's weighted aggregation across eight tasks.

As shown in Table 2, StarCore-9B achieves the highest overall reasoning score of 52.8, outperforming the second-best model, MiniMax M3 (47.7), by 5.1%. Notably, StarCore-9B achieves the best results in five evaluation dimensions, including security analysis (57.0), state completion (60.2), code understanding (76.4), agent trajectory understanding (42.2), and network traffic understanding (66.6). These results demonstrate its strong ability to reason about heterogeneous system artifacts and reconstruct the underlying system behaviors. In particular, its leading performance on state completion suggests that the model can effectively leverage contextual evidence to infer missing system states, while its performance on code and network traffic indicates its ability to establish meaningful connections between different types of system evidence.

We also observe that StarCore-9B does not achieve the highest score in every individual dimension. For example, DeepSeek-v4-Flash performs better on state prediction (46.3) and system command understanding (38.4). Nevertheless, StarCore-9B maintains the highest overall reasoning score, indicating a more balanced reasoning capability across heterogeneous system data and understanding tasks. This is particularly important for cybersecurity world models, where reliable reasoning requires integrating diverse system evidence rather than excelling at an isolated task.

Combining these results with the answer accuracy evaluation in Table 1, we find that StarCore demonstrates improvements in both final-answer correctness and reasoning correctness. The answer accuracy evaluation measures whether the model reaches the correct conclusion, whereas the reasoning evaluation examines whether the rationale supporting that conclusion is itself correct. The

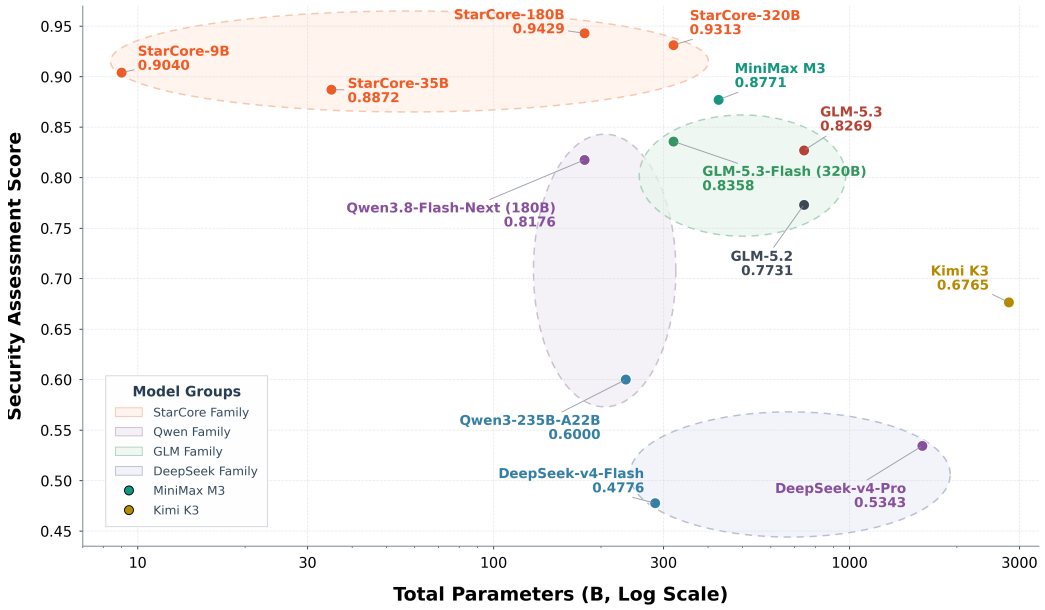


Figure 9: Relationship between total parameter count and security assessment accuracy. The horizontal axis uses a logarithmic scale. Shaded ellipses denote model families, while MiniMax M3 and Kimi K3 are shown as individual models.

complementary results provide evidence that state-chain learning improves not only the correctness of model outputs but also the quality of the underlying system-world reasoning process.

5.4 Model Scale Analysis

To further investigate whether stronger security assessment capability primarily arises from parameter scaling, we analyze the relationship between total parameter count and security assessment accuracy. As shown in Figure 9, the horizontal axis represents the total number of model parameters on a logarithmic scale, while the vertical axis reports accuracy on the security assessment task. Shaded ellipses group models from the same family, enabling a joint examination of scaling trends within individual families and differences across families. This analysis complements the overall results in Table 1 by focusing specifically on security assessment, which requires models to interpret system-state evidence, reason about behavioral dependencies, and derive reliable security conclusions.

Model Scale Alone Does Not Determine Security Assessment Accuracy. The results reveal no consistent monotonic relationship between parameter count and security assessment accuracy across the evaluated models. For example, Qwen3.8-Flash-Next achieves an accuracy of 0.8176 at the 180B scale, whereas Qwen3-235B-A22B obtains 0.6000 despite having a larger total parameter count. Similarly, Kimi K3, with 2,800B parameters, achieves 0.6765, remaining below several models that are substantially smaller. Within the DeepSeek family, increasing the model scale from DeepSeek-v4-Flash to DeepSeek-v4-Pro improves accuracy from 0.4776 to 0.5343, but the larger model still does not reach the performance of many smaller counterparts. These observations indicate that parameter scaling alone is insufficient to guarantee accurate interpretation of security-relevant system behaviors.

Intra-Family Differences Highlight the Role of Model Design and Training. Models within the same family also exhibit substantial variation at comparable or even identical scales. GLM-5.2 and GLM-5.3, both positioned at approximately 744B total parameters in our analysis, obtain security assessment accuracies of 0.7731 and 0.8269, respectively, corresponding to an absolute difference of 0.0538. Moreover, GLM-5.3-Flash achieves 0.8358 at the smaller 320B scale, exceeding both larger variants. The Qwen family shows an even more pronounced difference: the 180B Qwen3.8-Flash-Next outperforms Qwen3-235B-A22B by 0.2176 in absolute accuracy. These within-family differences suggest that backbone design, training data, post-training strategies, and alignment with

security reasoning tasks can substantially influence performance beyond the effect of parameter count.

StarCore Demonstrates Strong Accuracy–Scale Trade-offs. All four StarCore variants outperform the strongest non-StarCore model, MiniMax M3, which achieves an accuracy of 0.8771 at the 428B scale. Notably, StarCore-9B obtains 0.9040, exceeding MiniMax M3 by 0.0269 despite being approximately 47 times smaller in total parameter count. StarCore-35B also remains ahead of MiniMax M3 with an accuracy of 0.8872. These results demonstrate that state-chain learning can provide strong security assessment capability without relying on extremely large model scales. Although our evaluation does not directly measure inference latency or computational cost, the performance of the compact variants suggests a favorable accuracy–scale trade-off for deployment scenarios with limited computational resources.

StarCore Exhibits Non-Monotonic Scaling Behavior. Among the evaluated models, StarCore-180B achieves the highest security assessment accuracy of 0.9429, improving upon StarCore-9B by 0.0389. StarCore-320B also performs strongly with an accuracy of 0.9313, but remains 0.0116 below StarCore-180B. In addition, StarCore-35B does not surpass the 9B variant on this task. Therefore, the benefits of StarCore should not be interpreted as a simple consequence of increasing model size. Instead, the results suggest that state-chain supervised fine-tuning and reinforcement learning play a central role in establishing accurate security reasoning, while the final performance at each scale may also depend on the backbone, training configuration, and task alignment.

Overall, Figure 9 shows that explicit modeling of system-state dependencies can be more important for security assessment than parameter scaling alone. The strong performance of StarCore-9B and StarCore-35B supports the feasibility of compact cybersecurity world models, whereas StarCore-180B provides the highest accuracy among the evaluated variants. Nevertheless, these observations are specific to the security assessment task and should not be regarded as a universal scaling law. Together with the overall accuracy results in Table 1 and the reasoning evaluation in Table 2, they indicate that reliable cybersecurity world modeling requires both appropriate model capacity and explicit learning of system-world states, transitions, and security implications.

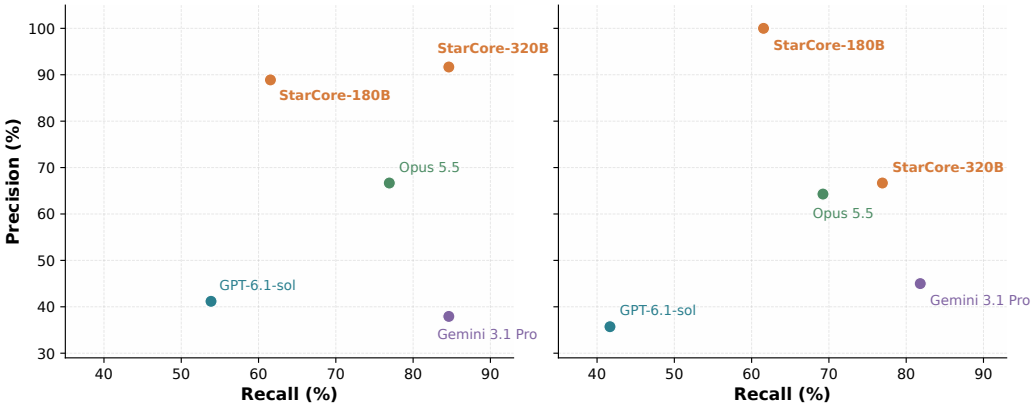
6 Capability Analysis and Case Study

To examine whether the system-world understanding capabilities of StarCore transfer to code-level security analysis, we conduct an exploratory study on a mixed collection of code snippets. The original collection contains 83 samples with reference answers but no independently verified vulnerability labels. We deterministically select 50 samples using a fixed hash-based rule, resulting in 13 reference-positive and 37 reference-negative cases. Five models are evaluated under two conditions: (1) *binary classification*, which requests only a vulnerable-or-benign decision, and (2) *evidence explanation*, which additionally requests the vulnerability mechanism, code evidence, assumptions, and limitations. The code and question are identical across the two conditions. All 500 expected model–prompt–sample combinations are recorded, and failed responses are retained in the availability analysis rather than replaced through selective rerunning.

Precision, recall, and F1 are computed over valid responses using the provided reference labels, whereas response availability is reported over all 50 samples. Consequently, these metrics measure agreement with the supplied references rather than verified performance on an independent vulnerability ground truth. This distinction is essential because the reference collection contains both directly inspectable risks and cases whose labels depend on missing caller, memory-safety, or runtime context.

Judgment with Binary Classification. Figure 10(a) shows that StarCore-320B provides the strongest binary classification result under the reference-label protocol. It detects 11 of the 13 reference-positive cases with only one reference false positive, yielding a precision of 91.67%, a recall of 84.62%, and an F1 score of 88.00%. StarCore-180B exhibits a more conservative operating point: its precision reaches 88.89%, but its recall is 61.54%, corresponding to eight detected reference-positive cases and one reference false positive. These two variants therefore illustrate different detection behaviors rather than a single monotonic scaling trend.

The baseline models occupy distinct regions of the precision–recall space. As shown in Table 3, Gemini 3.1 Pro [9] also detects 11 reference-positive cases, but produces 18 reference false positives and achieves only 37.93% precision. Opus 5.5 [1] provides a more balanced result, with 66.67%



(a) Judgment with binary classification.

(b) Judgment with evidence explanation.

Figure 10: Precision–recall comparison on the 50-sample code security set under (a) judgment with binary classification and (b) judgment with evidence explanation.

Table 3: Code security judgment results on the 50-sample code security set. Precision, recall, and F1 are computed over valid responses, while full-set agreement is calculated over all 50 samples.

Model	Valid Responses	Precision	Recall	F1	Full-Set Agreement	TP/TN/FP/FN
StarCore-180B	50/50	88.89%	61.54%	72.73%	88.00%	8/36/1/5
StarCore-320B	50/50	**91.67%**	**84.62%**	**88.00%**	**94.00%**	11/36/1/2
GPT-6.1-sol	50/50	41.18%	53.85%	46.67%	68.00%	7/27/10/6
Gemini 3.1 Pro	49/50	37.93%	84.62%	52.38%	58.00%	11/18/18/2
Opus 5.5	50/50	66.67%	76.92%	71.43%	84.00%	10/32/5/3

Table 4: Code vulnerability classification results under judgment with evidence explanation.

Model	Valid Responses	Precision	Recall	F1	Full-set Agreement	TP/TN/FP/FN
StarCore-180B	50/50	**100.00%**	61.54%	**76.19%**	**90.00%**	8/37/0/5
StarCore-320B	50/50	66.67%	**76.92%**	71.43%	84.00%	10/32/5/3
GPT-6.1-sol	48/50	35.71%	41.67%	38.46%	64.00%	5/27/9/7
Gemini 3.1 Pro	41/50	45.00%	81.82%	58.06%	56.00%	9/19/11/2
Opus 5.5	49/50	64.29%	69.23%	66.67%	80.00%	9/31/5/4

precision and 76.92% recall, whereas GPT-6.1-sol [19] obtains 41.18% precision and 53.85% recall. Overall, the binary condition demonstrates that StarCore can attain high agreement with the supplied references, particularly at the 320B scale. However, reference agreement alone does not establish whether the model has identified the correct vulnerability mechanism or whether the reference label itself is valid.

Judgment with Evidence Explanation. As shown in Figure 10(b), StarCore-180B increases its precision to 100% while leaving recall unchanged at 61.54%. The paired responses show that this change results from correcting one reference false positive, without detecting any additional reference-positive case. In contrast, GPT-6.1-sol and Opus 5.5 show mixed paired transitions rather than consistent gains.

As shown in Table 4, the two StarCore variants exhibit substantially different responses to this prompt condition. The baseline models likewise show no consistent benefit from explanation prompting. GPT-6.1-sol obtains only 35.71% precision and 41.67% recall, with its F1 score decreasing to 38.46%. Gemini 3.1 Pro retains relatively high conditional recall at 81.82% and improves its F1 score to 58.06%, but only 41 of its 50 responses are valid. Its full-set agreement therefore decreases slightly to 56.00%. Opus 5.5 remains comparatively balanced, achieving 64.29% precision and 69.23% recall, although its F1 score and full-set agreement decrease to 66.67% and 80.00%, respectively.

7 Conclusion

In this paper, we present StarCore, a cybersecurity world model that advances large language models toward dynamic understanding and reasoning of cyberspace. StarCore introduces a unified system state-chain representation to capture the temporal and behavioral relationships among heterogeneous data, including code execution, network traffic, system commands, and agent interactions. Building on this representation, we develop a two-stage learning framework that combines state-chain supervised fine-tuning with GRPO-based reinforcement learning, enabling the model to understand system states, reason about behavioral dependencies, predict potential attacks, and perform security assessments. Extensive experiments demonstrate that StarCore achieves strong performance across diverse cybersecurity tasks. In particular, StarCore-180B achieves an overall answer accuracy of 0.9238, outperforming the strongest baseline by 16.77 percentage points, while the compact StarCore-9B achieves the highest reasoning correctness score of 52.8 and surpasses much larger models in security assessment. These results highlight the effectiveness of state-chain modeling and learning for cybersecurity world models.

References

[1] Anthropic. Claude Opus 5.5. https://www.anthropic.com/claude-opus-5-5, 2026. Official model release, September 2026.

[2] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira. FlowLens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*. The Internet Society, 2021.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. In *Journal of Machine Learning Research*, volume 3, pages 1137–1155, 2003.

[4] BigModelGym. Cyberspace world model benchmark · security cognition in cyberspace. https://www.bigmodelgym.com/?lang=en#starcore, 2026.

[5] Cisco. Cisco AI for security. https://www.cisco.com/site/us/en/products/security/artificial-intelligence-ai/index.html, 2025.

[6] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[7] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

[8] C. Fu, Q. Li, and K. Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *NDSS*. The Internet Society, 2023.

[9] Google DeepMind. Gemini 3.1 Pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026. Published February 2026.

[10] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[11] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

[12] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de Las Casas, E. B. Hanna, F. Bressand, and et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[13] Kimi. Kimi api platform. https://platform.kimi.com/, 2025.

[14] Y. LeCun. A path towards autonomous machine intelligence. OpenReview, 2022.

[15] Microsoft. Cloud security solutions. http://microsoft.com/en-us/security, 2025.

[16] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: Real-time APT detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[17] MiniMax. MiniMax M3: Frontier coding, 1m context, native multimodality—all in one model. https://www.minimax.io/blog/minimax-m3, 2026. Official model release, June 2026.

[18] Mordor. Network security market size & share analysis source. https://www.mordorintelligence.com/industry-reports/network-security-market, 2025.

[19] OpenAI. GPT-6.1 Sol: A more capable sol across tasks. https://openai.com/index/introducing-gpt-6-1-sol/, 2026. Official model release, 2026.

[20] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[21] G.-. Team. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2018.

[22] M. A. Team. Llama 3. https://www.llama.com/models/llama-3/, 2025.

[23] Q. Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[24] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *NDSS*. The Internet Society, 2020.

Contribution

The listing of contributors is in alphabetical order based on their last names.

Core Contributors

Tianyu Cui	Ziwei Li	Weijie Wu	Ming Zhu
Miao Chen	Jiaxin Shi	Yue Xiao	Yixiang Zhang
Yang Li	Shouyi Wang		

Technique Advisors

Qi Li	Ke Xu

A Notation and Formal Problem Statement

This appendix consolidates the notation used throughout the paper and makes the distinction between latent system dynamics, observed artifacts, and model predictions explicit.

A.1 Notation

Table 5: Summary of notation.

Symbol	Meaning
t, T	Discrete time index and final index of an observed sequence.
o_t	Observed system artifact at time t, such as a code fragment, flow record, or command event.
a_t	Latent or annotated system action associated with a transition.
s_t	Canonical state representation used by the state-chain model.
$S = \{s_t\}_{t=0}^{T}$	A uni-modal or synchronized system state chain.
$\widetilde{S}$	A partially observed chain in which selected states are masked.
$\mathcal{M}$	Set of indices of missing states in a completion sample.
C	Environmental context, including the system or session context available to the model.
$Q, \mathcal{I}$	Task-specific question and system instruction.
τ	Task type: completion, prediction, or security assessment.
R	Action reasoning or an evidence-linked intermediate rationale.
Y	Final structured task output.
K	Prediction horizon measured in future states.
x	Complete model input $(\mathcal{I}, Q, S, C, \tau)$.
$o = (R, Y)$	A generated response trajectory; its meaning is distinct from an observation o_t.
$\pi_\theta, \pi_{\text{ref}}$	Trainable policy and frozen reference policy.
G	Number of responses sampled for one GRPO rollout group.
R_{SCR}	Aggregated State Chain Reward.

A.2 Observations, States, and Transitions

Let the latent evolution of a system be written as

$$s_{t+1}^{\star} \sim P_{\text{env}}\left(s_{t+1}^{\star} \mid s_t^{\star}, a_t\right), \qquad o_t \sim P_{\text{obs}}\left(o_t \mid s_t^{\star}\right), \tag{26}$$

where $s_t^{\star}$ denotes the generally unobserved true system state. StarCore does not assume that a single artifact fully determines $s_t^{\star}$. Instead, raw observations are normalized into model-facing states $s_t = \phi(o_{\leq t}, C)$, where ϕ denotes deterministic parsing, canonicalization, and context attachment. A state chain is therefore an evidence sequence rather than a claim of complete observability.

For training, the action a_t may be unavailable. In this case, the learning target is the observed transition regularity $P(s_{t+1} \mid s_{\leq t}, C)$, and action reasoning is treated as an explanation constrained by available evidence. When reliable action annotations exist, they can provide additional supervision, but they should remain distinguishable from model-generated rationales in the released data format.

A.3 Uni-Modal Chains and Interaction Graphs

A uni-modal chain of modality m is an ordered sequence

$$c^{(m)} = \left(s_0^{(m)}, s_1^{(m)}, \ldots, s_T^{(m)}\right), \qquad (s_t^{(m)}, s_{t+1}^{(m)}) \in E_{\text{temp}}, \tag{27}$$

where E_{temp} contains edges justified by execution order or event time. A multi-modal interaction graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ equal to the union of canonical states and

$$\mathcal{E} = E_{\text{temp}} \cup E_{\text{align}} \cup E_{\text{dep}}. \tag{28}$$

Here, E_{align} links observations of the same underlying event across modalities, while E_{dep} records explicitly supported behavioral dependencies. These edge types should not be conflated, i.e., temporal adjacency alone does not establish causality, and a cross-modal alignment edge does not by itself prove malicious intent.

B State-Chain Construction and Data Quality Control

This section gives a reference protocol for converting heterogeneous artifacts into state chains. It separates deterministic data transformations from semantic annotations and identifies the checks needed to make the resulting samples auditable. Implementation-specific thresholds, when used, should be recorded in dataset cards rather than hidden in preprocessing code.

B.1 Construction Pipeline

Algorithm 1 Reference state-chain construction procedure

Require: Raw artifact collection $\mathcal{D}_{\text{raw}}$ and source metadata
Ensure: Validated state-chain samples with provenance

1: Preserve source identifiers, collection time, licenses, and integrity hashes.
2: Parse each artifact into atomic events; retain unparsable records in a rejection log.
3: Canonicalize timestamps, identifiers, field types, and redacted values without discarding the source mapping.
4: Partition events into sessions using modality-specific chain keys.
5: Sort events by execution order when available, otherwise by timestamp with an explicit uncertainty flag.
6: Form uni-modal chains and attach the environmental context available at each cutoff.
7: Add cross-modal links only when shared identifiers, aligned sessions, or other documented evidence support them.
8: Validate ordering, schema conformity, target isolation, and train–test separation.
9: Return accepted chains, rejection reasons, and aggregate quality reports.

The rejection log is part of the data pipeline rather than disposable debugging output. It enables users to distinguish absence of malicious behavior from absence of usable observations. Similarly, redaction mappings should be stable within a chain so that entity consistency is preserved even when raw identifiers cannot be released.

B.2 Modality-Specific State Units

Canonicalization should remove serialization differences that are irrelevant to the task while preserving security-relevant distinctions. For example, equivalent whitespace can be normalized in structured records, but command arguments, exit status, traffic direction, and tool-call errors must not be discarded. When multiple raw events map to one canonical state, the mapping should be many-to-one and traceable rather than silently overwriting observations.

B.3 Temporal Alignment and Cross-Modal Linking

Clock skew and collection latency make exact timestamp equality an unreliable linking rule. A defensible alignment record should contain the linked state identifiers, the evidence used for linking, and an uncertainty indicator. Shared process or session identifiers provide stronger evidence than timestamp proximity alone. If only time windows are available, the link should be treated as uncertain and excluded from labels that require definitive cross-modal causality.

A cutoff-safe representation is essential for prediction. For a sample with cutoff t, the input construction function must satisfy

$$\text{Input}(t) = \text{Filter}\left(\mathcal{D}_{\text{raw}},\ \text{available_at} \le t\right), \tag{29}$$

and must exclude future flow summaries, post-incident annotations, final command outcomes, and agent reflections produced after t. This restriction prevents subtle label leakage even when the visible future states themselves are masked.

B.4 Missingness and Chain Integrity

Completion targets should be generated from intact source chains before masking. Let $M \in \{0,1\}^{T+1}$ denote a mask with $M_t = 1$ for a hidden state. The corrupted chain is

$$\widetilde{s}_t = (1 - M_t)s_t + M_t\ \texttt{[MASK]}, \qquad t \in \{0, \ldots, T\}. \tag{30}$$

Table 6: Reference verifier contract for State Chain Reward components.

Component	Required evidence	Typical checks	Failure isolated by the component
Format	Response and task schema	Parseability, required fields, valid indices, label vocabulary	Structurally invalid or incomplete output.
State understanding	Visible states and parsed response entities	Entity consistency, direction, event type, temporal references	Misreading an otherwise well-formatted observation.
Causal reasoning	Visible states, rationale, optional action annotations	Evidence coverage, transition consistency, unsupported inference	Correct label supported by an invalid explanation.
Attack prediction	Pre-cutoff input, predicted states, reference future	Horizon validity, future-state agreement, calibrated alternatives	Plausible narrative inconsistent with the actual future.
Security assessment	State evidence, rationale, final assessment	Label agreement, evidence sufficiency, benign alternatives	Unsupported attribution or missed malicious behavior.

Masks should be stored separately from the prompt so that evaluation can distinguish random, contiguous, and modality-specific missingness. Naturally missing observations should not automatically be treated as completion targets unless their values can be recovered from an independent source.

Chain validation covers four properties: (i) *well-formedness*, meaning that every state conforms to the schema; (ii) *order validity*, meaning that sequence indices and event times do not conflict; (iii) *referential validity*, meaning that cross-modal links resolve to existing states; and (iv) *target isolation*, meaning that no answer field is copied into an input field. Samples that fail these checks should be repaired deterministically or excluded with a logged reason.

C State Chain Reward and GRPO Implementation Details

This appendix expands the reward definitions in Section 4.2 into verifiable inputs, task-dependent activation rules, and optimization diagnostics. It does not introduce additional reward weights or training hyperparameters; those values should be reported with the corresponding training configuration.

C.1 Verifier Inputs and Failure Isolation

Each reward component should be computed from declared evidence rather than from an unconstrained judgment of the whole response. Table 6 lists a reference verifier contract. Programmatic checks are preferable for schema validity, state indices, and exact state reconstruction. Learned or human judges, when used for semantic components, should be calibrated separately and their agreement reported.

Let $v_k(x, o)$ denote the raw output of verifier k. A bounded reference normalization is

$$r_k(x, o) = \frac{\min(\max(v_k(x, o), l_k), u_k) - l_k}{u_k - l_k}, \qquad u_k > l_k, \tag{31}$$

where $[l_k, u_k]$ is the declared raw score range. If a verifier is already bounded in $[0, 1]$, no additional normalization is required. Scores from different verifier types should not be combined until their ranges and missing-value conventions have been made explicit.

C.2 Task-Dependent Reward Activation

For a task τ, the active set is $\mathcal{K}_\tau = \{k : m_{\tau,k} = 1\}$ and

$$\widetilde{w}_{\tau,k} = \frac{m_{\tau,k} w_k}{\sum_{j \in \mathcal{K}} m_{\tau,j} w_j}, \qquad R^{(\tau)}_{\mathrm{SCR}}(x, o) = \sum_{k \in \mathcal{K}} \widetilde{w}_{\tau,k} r_k(x, o). \tag{32}$$

If no component is active, the sample is configuration-invalid and should not enter training. A malformed response receives a low format score, but the treatment of its remaining components must

Table 7: Reference activation mask $m_{\tau,k}$. An entry of 1 denotes an applicable component; 0 denotes an inactive component. Active weights are renormalized for each task as in the main text.

Task	Format	State	Causal	Prediction	Assessment
Completion	1	1	1	0	0
Prediction	1	1	1	1	0
Security assessment	1	1	1	0	1

be fixed in advance. Assigning zero to all semantic components is stricter. Therefore, evaluating whatever can be parsed is more diagnostic. Mixing these conventions across runs would make reward comparisons difficult to interpret.

C.3 GRPO Update Procedure

Algorithm 2 One state-chain GRPO optimization step

Require: Policy π_θ, reference policy π_{ref}, batch $\mathcal{B}$, group size G
Ensure: Updated policy parameters and reward diagnostics
for each input $x \in \mathcal{B}$ **do**
  Sample $\{o_i\}_{i=1}^{G}$ from the rollout policy $\pi_{\theta_{\text{old}}}$.
  Evaluate active SCR components and store component-level scores.
  Compute $r_i = R_{\text{SCR}}(x, o_i)$ for every candidate.
  Compute μ_r and σ_r within the rollout group.
  if σ_r is below the declared numerical threshold **then**
    Set group advantages to zero and mark the group as degenerate.
  else
    Compute $\hat{A}_i = (r_i - \mu_r)/(\sigma_r + \epsilon)$.
  end if
end for
Maximize the clipped objective with KL regularization over valid groups.
Log reward distributions, format failures, KL statistics, and degenerate-group rates.

A degenerate group contains candidates with identical or nearly identical rewards. Its relative advantages carry no useful ordering signal. We set them to zero to avoid amplifying numerical noise. The cause should still be inspected. Repeated degenerate groups may indicate that prompts are too easy, sampling diversity is too low, or the verifier cannot distinguish meaningfully different responses.

C.4 Reward Diagnostics and Anti-Gaming Checks

A scalar training reward can rise while one underlying capability deteriorates. Component-level curves should therefore be reported alongside the aggregate SCR. Useful diagnostics include format validity, average evidence coverage, future-state accuracy, assessment accuracy, response length, KL divergence, and the fraction of degenerate groups. Abrupt changes in only one component should trigger a verifier audit before being interpreted as improved world understanding.

The verifier itself should be tested with counterexamples: a correct answer paired with fabricated evidence; a well-formatted response that cites invisible states; a prediction copied from post-cutoff context; a benign chain labeled malicious without support; and a verbose rationale that repeats observations without explaining a transition. These probes help detect reward hacking and distinguish structural validity from semantic correctness. Human review should focus on samples where the answer and rationale rewards disagree, since those cases expose failures hidden by final-answer accuracy alone.