Fuzzing:Multi-objective optimization

Zhentao Zhu, Zhenyu Wen,

***Abstract*—Height estimation (HE) from high-resolution remote sensing imagery is crucial for 3D scene understanding, with applications such as urban planning and 3D reconstruction. Meanwhile, semantic segmentation (SS) identifies categories and content within these images. Integrating HE and SS can generate more accurate object models, underscoring the need for a unified multi-task framework. Current approaches perform HE and SS independently, overlooking their potential synergy. Therefore, we propose a multi-task network (HEASSNet) to explore the correlation between HE and SS and achieve collaborative learning. Specifically, the two tasks share a pre-trained SAM encoder. We introduce a Low-Rank Adaptation fine-tuning strategy with convolution operations and enhance the local prior in the SAM encoder. The SS branch uses Swin-Transformer to capture global context for qualitative analysis and classification. The HE branch uses convolutional blocks that capture local features to perform quantitative analysis and generate height maps. To learn the distribution of height values, we propose a Feature Enhancement and Aggregation Gate to fuse local and global features, integrating the global context of SS into HE. In addition, the generated height maps provide valuable prompts to the SS branch, distinguishing easily confused categories. Extensive evaluations across multiple datasets validate the effectiveness of our approach.**

***Index Terms*—Semantic Segmentation, Height Estimation, Multi-Tasks Learning, Low-Rank Adaptation.**

I. Introduction

HEIGHT estimation (HE) and semantic segmentation (SS) are fundamental yet challenging tasks in the intelligent interpretation of remote sensing images (RSI). They are critical for various applications, including 3D mapping, urban planning, agricultural management, and forest monitoring. Recent advancements in deep learning have significantly enhanced the performance of both HE and SS. However, the development of remote sensing technologies has introduced the need to tackle complex problems using multi-modal (data or tasks). Despite the individual progress in HE and SS, their potential synergy has often been overlooked. HE provides 3D spatial information, while SS contributes to semantic understanding and component recognition. Each task offers unique perspectives on scene parsing and can implicitly or explicitly inform the other. Consequently, joint learning of HE and SS within a unified network is a promising avenue for research.

Yachen Wang, Yejian Zhou and Huayong Tang are with College of Information Engineering, Zhejiang University of Technology, Hangzhou, 310023, P.R.China.

Guanyong Wang is with the School of Information and Science, North China University of Technology, Beijing 100144, China.

Lei Zhang is with the School of Electronics and Communication Engineering, Sun Yat-sen University, Shenzhen 518107, China.

The paper is supported by National Natural Science Foundation of China (Grant No. 62471438 and 62401018), the Zhejiang Provincial Natural Science Foundation of China (Grant No. LY23F010012). (Corresponding author: Yejian Zhou, email: yjzhou25@zjut.edu.cn;)

Driven by advancements in deep learning, HE and SS have achieved significant progress in remote sensing. For HE, Ghamisi et al. [1] proposed IMG2DSM, a method based on Generative Adversarial Networks (GANs) with skip connections to predict Digital Surface Model (DSM). Paoletti et al. [2] utilized Variational Autoencoders (VAEs) and GANs to generate DSM from single optical images. For SS, significant research has made progress in land cover classification, agricultural monitoring, and urban scene recognition [3]–[5]. Despite these successes, HE and SS have developed independently as separate research. This separation limits the full potential of intelligent interpretation of remote sensing. To address this limitation and enable joint learning of HE and SS, it is necessary to design a multi-task learning (MTL) framework.

The MTL networks are developed to improve performance by leveraging the inherent correlations between tasks. In computer vision, MTL has demonstrated feasibility across various tasks, including SS and depth estimation, SS and object detection, and object detection and classification [6]–[9]. Similarly, in the remote sensing community, MTL has shown promising progress. To extract buildings across various scenes, Guo et al. [10] developed a multi-task framework integrating scene classification and pixel mapping. Feng et al. [11], leveraging computational imaging techniques, proposed a network for super-resolution and colorization. Additionally, substantial advancements have been made in SS and change detection (CD) tasks [12], [13]. Despite the clear relevance and consistency between HE and SS, the potential for joint learning of these tasks remains largely unexplored, resulting in suboptimal results.

In addition, while MTL is a solution, a significant challenge lies in the potential interference among tasks, which may disturb parameter updates and degrade overall performance. The Segment Anything Model (SAM) [14] is a large-scale vision model trained on extensive datasets, offering robust feature extraction capabilities. Leveraging its pre-trained encoder to extract shared features can effectively address this issue. However, SAM's training data is primarily natural images, and its applicability to RSI remains underexplored. Furthermore, SAM is based on Transformer architecture, which relies on global attention mechanisms. Given that RSI contains numerous small-scale objects, there is a need for the network to possess strong local attention capabilities.

Inspired by recent advancements in MTL and SAM, we propose HEASSNet, an MTL network designed to achieve HE and SS from optical RSI. HEASSNet avoids relying on specific optimization strategies for HE and SS, choosing a simple architecture with reduced complexity. Specifically, we use the pre-trained encoder of SAM and propose a LoRA fine-tuning method with convolution to address the local limitation of

SAM. HE and SS are executed in independent branches. The SS branch uses the swin transformer (SwinT), which excels at capturing long-range dependencies and ensuring accurate judgments in detail and overall. The HE branch consists of convolutional blocks, which effectively obtain details and make accurate predictions but are limited in capturing global features. For this, we propose a feature enhancement and aggregation gate (FEA-Gate). The module enhances local and global features, selectively aggregating this information into the HE branch, improving its global perception and contextual information. The height result is input into the prompt encoder and is summed pixel-wise with the features of the SS branch, effectively distinguishing confused categories and improving segmentation accuracy.

In summary, our contributions are summarised as follows:

- We propose HEASSNet, an MTL network designed to interpret high-resolution RSI through SS and HE. HEASSNet leverages joint learning to constrain and optimize performance, achieving state-of-the-art results on diverse datasets.
- We propose a fine-tuned method based on LoRA. Using convolution operations for multi-scale features in LoRA, solving the limitations of the encoder and enhancing the local prior, while retaining the strong generalization of SAM, making it more suitable for RSI.
- To achieve a positive transfer of favorable information in cross-task, we propose the FEA-Gate. The module effectively filters out irrelevant and redundant features from various modalities, allowing the incorporation of sufficiently complementary information to form a comprehensive representation for predicting height value.

Organization. In Section II, we review the existing work closely related to ours. We propose the details of HEASSNet Section III. The discussion of experiments is given in Section IV. In Section V, we conclude our paper.

II. Background and Motivation

In this section, we introduce the primary technical concepts of protocol fuzzing and clarify the main challenges we aim to address in this paper.

A. Protocol Fuzzing

To facilitate standardized communication across networked systems, the Internet Engineering Task Force (IETF) established technical specifications through Requests for Comments (RFCs). These documents define protocol architectures and operational requirements, as exemplified by the File Transfer Protocol (FTP) standardized in RFC 959. Protocol implementations rely on well-defined message structures and stateful interactions. As illustrated in Fig.1, FTP messages comprise three core components: a message command type, structured headers with key-value parameters, and carriage return and line feed characters (CRLF). Fig.2 further demonstrates the state machine governing protocol progression, where implementations transition from the INIT state to the AUTH state upon successful authentication using USER and PASS commands. Subsequent advancement to the TRAN state requires processing additional message types beyond basic authentication credentials.

Fuzz testing tools systematically generate message sequences to evaluate protocol robustness, ideally producing syntactically valid inputs that follow specified state transitions. Effective test case generation must account for both message structure validity and state-dependent sequencing constraints to accurately validate protocol conformance. This structured approach enables comprehensive verification of implementation adherence to RFC specifications while identifying potential edge-case vulnerabilities.

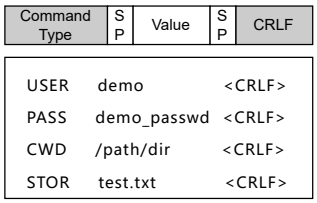

Fig. 1: FTP command structure and an example of FTP request from Lightftp.

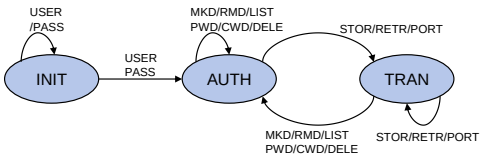

Fig. 2: FTP state model.

B. Task Scheduling

The protocol fuzzing framework (Fig.3) employs a modular architecture with three core phases: **test case generation, execution feedback, and vulnerability feedback.** The test case generation phase, as the central module, constructs specification-driven inputs through a three-tier mechanism: 1) Defining protocol state transitions and message syntax via communication modeling ; 2) Optimizing test sequence prioritization using task scheduling algorithms to target critical states and edge-case paths; 3) Generating RFC-compliant test cases that ensure syntactic validity and state-machine consistency. This phase achieves dynamic adaptation of protocol rules through co-evolution of seed libraries and input models.

The remaining phases leverage dynamic injection and anomaly detection for state monitoring and defect identification, forming a closed-loop validation system. The framework ensures systematic exploration of protocol implementations through its layered, specification-aware design.

The core focus of this study lies in the task scheduling algorithm, which employs a hierarchical mechanism to decouple fuzzing into two phases: (1) Inter-state scheduling: Selecting protocol states for testing based on state priority metrics; (2) Intra-state scheduling: Generating targeted test cases within

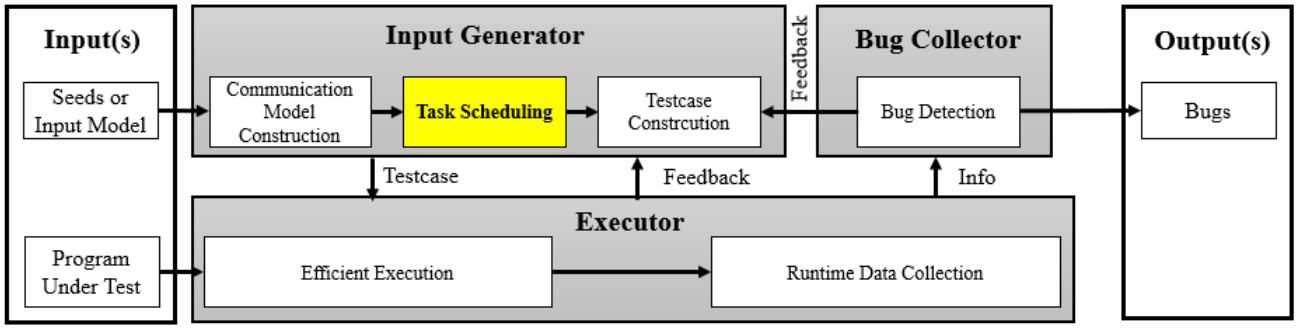

Fig. 3: Summarized workflow of existing protocol fuzzers.

selected states using conventional methods (e.g., seed scheduling, byte scheduling, mutation strategy scheduling). This architecture achieves fine-grained control through three heuristic strategies: Rarity-preferred [19, 110, 123, 162] prioritizes under-explored states to uncover latent logic; Performance-preferred [30, 55, 56, 110, 123] emphasizes states with high code coverage or defect discovery rates; Complexity-preferred [45, 57] selects structurally complex or deep states (e.g., ICS3Fuzzer [45] prioritizes deep states, Pulsar [57] employs mutable field weight calculations).

However, since all these state selection algorithms are implemented and evaluated separately on different platforms and targets, it is difficult to make a fair comparison and achieve conclusive findings. Liu et al [85] evaluate the three existing state selection algorithms of AFLNET [123], including a rarity-preferred algorithm, an algorithm that randomly selects states, and a sequential state selection algorithm. They find that these algorithms achieved very similar results in terms of code coverage. They attribute the reasons to the coarse-grained state abstraction of AFLNET and the inaccurate estimation of the state productivity. Therefore, they propose the AFLNETLEGION algorithm [85] to address these issues, which is based on a variant of the Monte Carlo tree search algorithm [84].

	INIT	AUTH	TRAN
INIT	2	1	0
AUTH	0	6	3
TRAN	0	6	3

Fig. 4: FTP state transition out-degree and in-degree matrix diagram.

C. Motivation

Existing fuzzing strategies for stateful protocol implementations exhibit critical limitations: (1) **Short-sighted testing approaches constrain exploration of deep-state execution paths.** Conventional approaches prioritize minimizing message interactions by favoring shorter viable sequences for protocol state transitions. While optimizing resource efficiency, this strategy overlooks the cascading effects of intermediate state transitions. For instance, in FTP implementations, reaching the TRAN state can be achieved through multiple command sequences (e.g., [PASS, USER, STOR], [PASS, USER, LIST, STOR], or extended sequences like [PASS, USER, CWD, LIST, MKD, STOR]). Though longer sequences incur higher overhead from auxiliary operations (e.g., directory creation via MKD), these operations fundamentally modify the server's internal state, enabling richer feedback for subsequent test inputs.

Nevertheless, prevailing methods disproportionately favor shorter sequences, discarding semantically complex paths that may expose subtle vulnerabilities. Furthermore, protocol state transitions exhibit heterogeneous complexity levels (as shown in Fig. 4), where the AUTH-to-TRAN transition involves significantly more paths than INIT-to-AUTH transitions. This observation reveals the insufficiency of state coverage metrics alone for guiding effective testing. Current tools' neglect of intricate long-sequence patterns restricts their ability to exercise critical edge-case scenarios, thereby limiting deep-state vulnerability detection. To address this gap, this work proposes a state-aware testing methodology that systematically prioritizes path diversity over conventional state coverage metrics, thereby advancing vulnerability discovery in complex protocol implementations.

(2) **Existing grey-box fuzzing approaches for stateful network protocols exhibit significant limitations in their evaluation methodology.** While existing approaches acknowledge the importance of state selection and implement heuristic algorithms (e.g., RANDOM, ROUND-ROBIN, FAVOR), their oversimplified evaluation criteria inadequately capture the intrinsic value of state sequences. Current methods predominantly prioritize defect detection metrics at the expense of operational efficiency and favor minimal-path sequences rather than exploratory alternatives with higher diversity potential.To address these shortcomings, this study proposes a dual-objective optimization framework that simultaneously considers both execution time and bug discovery rate. Consequently, the current evaluation methodology exhibits limited capability in precisely quantifying the intrinsic value of discrete protocol states and their associated transition sequences, thereby adversely affecting the fuzzer's overall testing efficacy.

TABLE I: Main Parameters

Parameter	Sign	Content
Finite Set of States	Q	$q_1, q_2, \ldots, q_n$
Initial State of The System	q_0	/
Input Alphabet	I	$\sigma_1, \sigma_2, \ldots, \sigma_m$
Output Alphabet	Λ	$o_1, o_2, \ldots, o_n$
State Transition Function	δ	$\delta(q, \sigma) = q'$
Set of Outputs for State Transition	λ	$Q \times I \rightarrow \Lambda$
State-sequence Instance Pairs	C	$C_1, C_2, \ldots, C_l$

III. Method and Implementation

In this section, we first model the system using a finite-state machine and define our optimization objectives. Subsequently, we outline the details of the key components, including .

A. Problem Definition

In protocol fuzzing, state transitions often correlate with specific event occurrences. Finite State Machines (FSMs) provide a formal framework to characterize system behaviors by defining discrete states and their transition rules, making them particularly effective for modeling event-driven systems such as protocol implementations. By formalizing protocol logic into FSMs, explicit mappings between input sequences and state transitions can be established. These structured mappings guide fuzzing tools in generating context-aware test cases that probe deep program states and validate transition-path security. Key parameters used in this framework are summarized in Table I.

A Mealy FSM is a six-tuple:

$$\mathcal{A} = (Q, q_0, I, \Lambda, \delta, \lambda) \tag{1}$$

where $Q = \{q_1, q_2, \ldots, q_n\}$ represents a finite set of states, q_0 represents the initial state of the system, $I = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is the input alphabet, and $\Lambda = \{o_1, o_2, \ldots, o_n\}$ is the output alphabet; $\delta : Q \times I \rightarrow Q$ represents the state transition function and can be expressed as $\delta = \{(q, \sigma, q') : \delta(q, \sigma) = q'\}$. $\lambda : Q \times I \rightarrow \Lambda$ represents the set of potential outputs or observations for this transition.

Meanwhile, let I^* represents the set of finite-length sequences on I, and let Λ^* represents the state transition outputs over Λ. In this regard, by successive iterations, for any $q_0 \in Q$, δ can be extended over a k-length string $s_k = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_k} \in I^*$ by $\delta(q_0, s_k) := \delta(\delta(\ldots\delta(\delta(q_0, \sigma_{i_1}), \sigma_{i_2})\ldots), \sigma_{i_k})$. Also the output function λ can be extended over a k-length string $p_k = o_{j_1}o_{j_2}\ldots o_{j_k} \in \Lambda^*$ by $p_k = \lambda(q_0, s_k) := \lambda(\lambda(\ldots\lambda(\lambda(q_0, \sigma_{i_1}), \sigma_{i_2})\ldots), \sigma_{i_k})$.

Therefore, given a Mealy FSM $\mathcal{A}$, there exist different input-output pairs (Δ_i, O_j) composed of sequences $(\sigma_{i_1}, o_{j_1})(\sigma_{i_2}, o_{j_2})\ldots(\sigma_{i_k}, o_{j_k})$, where the state transition function $\delta(q_0, s_k)$ and the output function $\lambda(q_0, s_k)$ are defined over different existing sequences of events $s_k = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_k}$, ensuring that $|\delta(q_0, s_k)| > 0$ and $|\lambda(q_0, s_k)| > 0$, respectively.

For any protocol implementation, a corresponding Mealy FSM $\mathcal{A}$ can be constructed. During each round of fuzzing, the fuzzing tool $\mathcal{F}$ selects appropriate *state-sequence instance pair* $C = (q_i, s_k^j)$ for testing, where $q_i \in Q$ represents the target state to be tested in the current round. Let m_q denote the number of seed sequences (non-fixed and state-dependent) that can reach q_i, from which one seed sequence s_k^j is selected for testing.

Formally, if the protocol implementation triggers state-sequence instance pair C resulting in a crash, the crash type is recorded. This process iterates until reaching the maximum execution time limit. The experimental objective is to maximize the number of crashes within the limited time.

B. Optimization Goal

In fuzzing, we aim to maximize the number of discovered crashes (M) while maintaining execution time below the predefined threshold $T_{\max}$. Our formulation for the optimization objective reads:

$$\begin{aligned} &\arg\max M \\ &\text{s.t.} \quad T \leq T_{\max} \end{aligned} \tag{2}$$

First, define the non-negative integer variable x_{q,s_k} to represent the number of times the state sequence pair (q, s_k) is selected, where q represents a decision option from the candidate choice set Q, and s_k represents a seed sequence from the option q's corresponding seed sequence set m_q, with the constraint $x_{q,s_k} \in \mathbb{Z}^+$.

$$x_{q,s_k} \in \mathbb{Z}^+ \quad \forall q \in Q,\ s_k \in m_q \tag{3}$$

Crash Model. In this section, we aim to model the crashes M obtained through fuzzing. We define M as the cumulative crashes across all rounds generated by each state-sequence instance pair C in the fuzzing process:

$$M = \sum_{q=1}^{n} \sum_{s_k \in m_q} \sum_{p=1}^{x_{q,s_k}} \left(M_{q,s_k}^{E(p)} + M_{q,s_k}^{N(p)} \right) \tag{4}$$

where M_{q,s_k}^{E} represents crashes caused by protocol violations, M_{q,s_k}^{N} represents crashes during specification-compliant execution.

Execution Time Model. We define the execution time T during testing as the sum of execution times for each state-sequence instance pair C:

$$T = \sum_{q=1}^{n} \sum_{s_k \in m_q} \sum_{p=1}^{x_{q,s_k}} t_{q,s_k}^{(p)} \tag{5}$$

where t_{q,s_k} represents the total execution time summed across all rounds for each state-sequence instance pair.

Value Model. To accurately evaluate the exploration potential of each state-sequence instance pair and ensure the selection of higher-value pairs in subsequent fuzz testing, thereby maximizing crash discovery efficiency within limited execution time, we propose a novel value assessment model E_{q,s_k}. This model quantitatively estimates the exploration value (encompassing both crash count and execution time) of instance pairs to optimize testing performance:

$$E_{q,s_k} = \sum_{p=1}^{x_{q,s_k}} E_{q,s_k}^{(p)} \tag{6}$$

where E_{q,s_k} represents the total exploration value summed across all $selected\_time$ experimental rounds for each state-sequence instance pair C, calculated as the accumulation of $E^{(p)}_{q,s_k}$ values obtained in each round. The exploration value per experiment $E^{(p)}_{q,s_k}$ is defined as:

$$E^{(p)}_{q,s_k} = f^{(p)}_{\text{crash}}(q, s_k) \times f^{(p)}_{\text{time}}(q, s_k) \tag{7}$$

$$f^{(p)}_{\text{crash}}(q, s_k) = 2^{\log(P^{(p)}_{q,s_k} + \alpha_p \cdot M^{E(p)}_{q,s_k} + M^{N(p)}_{q,s_k} + \epsilon)} \tag{8}$$

$$\alpha_p = \frac{M^{E(p)}_{q,s_k}}{M^{E(p)}_{q,s_k} + M^{N(p)}_{q,s_k} + \epsilon} \tag{9}$$

The proposed state value evaluation formula consists of two components. **The first component is the crash discovery value, as formulated in Equation (8).** This metric is adapted from AFLNET's state value estimation formula, with modifications introduced to optimize crash count maximization under our testing framework. This component integrates the following key factors: (1) code path coverage ($P^{(p)}_{q,s_k}$), where greater coverage indicates higher probability of discovering potential vulnerabilities; (2) protocol-violation crashes ($M^{E(p)}_{q,s_k}$), whose impact is reduced through a weighting coefficient α_p since such crashes typically cannot trigger genuine vulnerabilities; α_p represents the proportion of $M^{E(p)}_{q,s_k}$ among the total number of crashes and ϵ is a smoothing factor ($\epsilon = 1$) to prevent division by zero; and (3) valid protocol crashes ($M^{N(p)}_{q,s_k}$), which exhibit higher vulnerability detection value as they conform to protocol specifications while triggering abnormal execution. The logarithmic relationship in this factor demonstrates a negative regulatory effect on exploration value as testing iterations increase, thereby effectively preventing excessive retesting of low-efficiency regions and improving the overall efficiency of fuzz testing.

$$f^{(p)}_{\text{time}}(q, s_k) = \beta_p \times \left(2 \log_{10}\left(\log_{10}(t^{(p)}_{q,s_k} + \epsilon)\right)\right)^{-1} \tag{10}$$

$$\beta_p = \beta_{\max} \times \left(1 - \sigma\left(\gamma\left(T_p - T_0\right) h^{-1}\right)\right) \tag{11}$$

The second part quantifies execution efficiency of state sequence instance pairs, as formulated in Equation (10). In order to accurately characterize the dynamic variation of path coverage and crash discovery efficiency over time during the fuzzing process, this study introduces a time-weighted dynamic reward mechanism. Experimental observations reveal that the rate of new path discovery and crash detection per unit time is significantly higher during the initial phase of fuzzing, indicating a pronounced time-dependent feature. To address this phenomenon, we propose the incorporation of a time gain function, which provides progressive reward compensation for exploration activities in later stages, thereby effectively mitigating the issue of diminishing returns in fuzz testing. Specifically, we define a dynamic weighting factor β_p that evolves with the cumulative execution time, formally expressed in Equation (11).

In this formulation, $\sigma(x)$ represents the standard Sigmoid function. The parameter $\beta_{\max}$ is used to set the maximum reward value during the early stages, ensuring that the fuzzer actively explores new paths and crashes at the beginning of the testing process. T_p represents the cumulative execution time of the ongoing fuzzing process, which dynamically tracks the progress of the testing. T_0 represents the temporal inflection point at which the reward begins to decay, guiding the fuzzer towards deeper exploration as the testing progresses. The conversion coefficient h = 60 min/h (minutes/hour) enables standardized temporal unit conversion, based on the SI definition where 1 h = 60 min, ensuring dimensional consistency in experimental calculations. The parameter γ controls the rate of reward decay, ensuring a smooth and gradual reduction in practice. Through this design, β_p maintains a high reward value in the early testing stages to accelerate the discovery of paths and crashes. As the testing progresses, the reward gradually converges, directing resources towards inputs with higher exploration potential. This dynamic adjustment mechanism effectively balances exploration and exploitation, significantly enhancing the efficiency of fuzz testing. These parameters will be optimized in subsequent experiments.

Meanwhile, to optimize the execution time of individual test cases, the model incorporates a time penalty factor with negative derivative properties (corresponding to the right-hand side of the equation). This design is theoretically grounded in two fundamental observations: prolonged execution time linearly reduces testing throughput, while execution efficiency demonstrates significant positive correlation with potential exploration value. The dual-regulation mechanism, integrating time-gain rewards and execution-time penalties, enables optimal resource allocation across both temporal and test-case dimensions through dynamic value assessment. This approach not only adheres to the fundamental exploration-exploitation trade-off principle in fuzz testing but also adaptively prioritizes high-potential test cases, thereby significantly improving overall testing efficiency. Where $t^{(p)}_{q,s_k}$ denotes the execution time of the current test cycle, whose value is determined based on the theoretical framework of AFLNet's state evaluation formula and subsequently validated through experimental studies.

C. MAB

This study adopts the Multi-Armed Bandit (MAB) algorithm as the core optimization strategy, dynamically balancing the trade-off between exploration and exploitation to select the optimal combination from candidate test case state sequences, with the aim of maximizing the objective function. The algorithm effectively improves optimization efficiency while maintaining solution quality. A detailed discussion on the theoretical foundations of the MAB algorithm and its applicability to this study will be provided in the following sections.

The Exploration-Exploitation Trade-off. The balance between exploration and exploitation is a fundamental problem in game theory. Inspired by the decision-making process of players seeking to maximize rewards from slot machines, the MAB model was proposed to address such optimization challenges.

In a classical MAB setting, there are N parallel arms, and only one arm can be selected at each round. The expected reward of arm i ($i \in \{1, 2, \ldots, N\}$) is represented as R_i. The objective of the MAB problem is to maximize the cumulative reward within a limited number of selections. However, the reward of an arm remains unknown until it is actually pulled. The process of **exploration** involves selecting different arms to gain more accurate estimates of their rewards, whereas once sufficient knowledge is obtained, choosing the arm with the highest expected reward corresponds to **exploitation**. In the short term, exploitation yields the highest immediate reward, while in the long term, systematic exploration can uncover better options and thus lead to greater overall returns. Therefore, effectively managing the exploration-exploitation balance is crucial to achieving optimal decision-making performance in MAB problems.

In the experiment, the system comprises n states, where each state q is associated with a set of reachable seed sequences m_q.

$$a_{i,j} \equiv C_l, \quad C_l \in \mathbf{C} \tag{12}$$

$$N = \sum_{q \in Q} |m_q| \tag{13}$$

Each state-seed sequence pair represents a fuzzing operation unit, i.e., a bandit arm $a_{i,j}$, resulting in a total of N distinct bandit arms, where N is the sum of reachable seed sequences across all states.

The reward $R(C_l, n_l)$ of each bandit arm $a_{i,j}$ is defined as E^{q,s_k}, representing its expected fuzzing effectiveness. In the algorithm design, the parameter n_l represents the cumulative selection count of state-sequence instance pair C_l (where $n_l \equiv x_{q,s_k}$, with x_{q,s_k} denoting the original counting variable). For notational simplicity, we consistently adopt n_l as the standardized representation throughout this paper.

$$R(C_l, n_l) = E_{q,s_k} = \sum_{p=1}^{n_l} E_{q,s_k}^{(p)} \tag{14}$$

Next, we can calculate a final score for the combinations to make decisions. UCB1 [27] is a classic answer to the MAB problem, and we calculate scores based on it as

$$\begin{aligned} \text{Score}(C_l, n_l) &= \overline{R}(C_l, n_l) + U(C_l, n_l) \\ &= \frac{\sum_{p=0}^{n_l} R(C_l, p)}{n_l} + \gamma \cdot \sqrt{\frac{\ln\left(\sum_{C_l \in C} n_l\right)}{n_l}} \end{aligned} \tag{15}$$

In the fuzzing framework, $\mathbf{C}$ denotes the set of all instance pairs. The scoring function $\text{Score}(C_l, n_l)$ consists of two components:

Where $\overline{R}(C_l, n_l)$ represents the average reward of pair C_l over n_l previous rounds and reflects historical performance for exploitation. $U(C_l, n_l)$ represents the upper confidence bound component. The algorithm begins with a **pioneering phase** that enforces $\forall C_l \in C, n_l = 1$. In subsequent rounds, the system dynamically selects:

$$C_{\text{next}} = \underset{C_i \in C}{\text{argmax}}\ \text{Score}(C_i, n_i) \tag{16}$$

The hyperparameter γ controls the exploration-exploitation trade-off.

D. Algorithm Design

Algorithm 1 The Fuzzobj algorithm

Input : P_t: Protocol Implementation
s_{k0}: Initial Seed Queue with state q_0
d: Dictionary Corpus
D: Current fuzzing depth
Output: M: Crash sequence count

1 Initialize FSM S and Seed Queue s_{k0} with q_0;
2 **Pioneer Phase:** Each C_l is selected once to initialize all $n_l \leftarrow 1$; After each fuzzing iteration, compute $SWVA\text{-}UCB(C_l, n_l)$ to guide future selection;
3 **repeat**
4 **State-sequence Pair Selection Phase:**
foreach $C_l \in C$ **do**
SWVA-UCB$(C_l, n_l) \leftarrow$ ComputeScore(C_l, n_l)
$\mathcal{K} \leftarrow$ GetTopKPairs(C, K) ; `// Rank by SWVA-UCB score`
$C_l^* \leftarrow$ BoltzmannSelect$(\mathcal{K})$
Energy Allocation Phase:
$E \leftarrow$ AssignEnergy(C_l^*)
Mutation Testing Phase:
for $i \leftarrow 1$ **to** E **do**
$INIT \leftarrow$ CalculateDepth(D, R)
if $NotINIT$ **then**
$s'_k \leftarrow$ StateGuidedMutate(s_k, d, q)
else
$s'_k \leftarrow$ RandomMutate(s_k, q)
if $ContextAware(s'_k, d)$ **then**
$M_E \leftarrow M_E \cup \{s'_k\}$
$R' \leftarrow$ SendToServer(P_t, s'_k)
if $IsCrash(s'_k, P_t)$ **then**
$M_N \leftarrow M_N \cup \{s'_k\}$
else if $IsInteresting(s'_k, P_t, q)$ **then**
$m_q \leftarrow m_q \cup \{s'_k\}$
$Q \leftarrow$ UpdateFSM(Q, R')
23 **until** *timeout* T ***or*** *abort-signal*;
24 **return** $M = M_E + M_N$

1) Overview: To enhance algorithmic dynamism and mitigate errors induced by fuzzing randomness, we propose Fuzzobj, an optimized variant of UAB1, with the following workflow: The proposed state-guided protocol fuzzing framework takes four key inputs: the target protocol implementation Pt, an initial sequence s_{k0} containing the starting state q_0, dictionary corpus d, and the fuzzing depth D. The primary output is the total number of crashes M detected during testing.

The fuzzing process begins with an initialization and pioneer phase (Lines 1-2), where all state-sequence pairs undergo preliminary testing. During the state-sequence pair

selection phase, the fuzzer first computes the SWVA-UCB score for each state-sequence pair (Lines 4-5), then constructs a Top-K candidate pool based on these scores (Line 6). The BoltzmannSelect algorithm is subsequently employed to select the target state-sequence pair Cl from this pool (Line 7), effectively guiding the fuzzer's exploration of the target server. The energy allocation phase dynamically distributes fuzzing resources according to historical testing results and selection frequencies (Line 8). In the mutation testing phase, the framework applies different fuzzing strategies to mutate sequence s_{k0} based on both the protocol's program state and current fuzzing depth (Lines 10-14). Following mutation, each sequence undergoes validity verification through a content-aware module (Lines 15-16). Sequences causing protocol specification violations are recorded as protocol errors (M_E) and immediately proceed to the next iteration. Valid sequences are sent to P_t for execution (Line 17). The fuzzer maintains two categories of mutated sequences s'_k : those that trigger crashes in P_t, and those that improve code or state coverage. For the latter case, the system correspondingly updates the Finite State Machine (FSM). This iterative process continues until the allocated fuzzing resources for the current round are exhausted, at which point a new state-sequence pair is selected for subsequent testing. The final output M represents the cumulative count of valid crashes detected throughout the testing campaign.

2) Detailed Techniques of ObjFuzz: **Dynamic Optimization of Evaluation Bias.** To address the evaluation bias arising from dynamic variations in path and crash rewards during fuzz testing, this paper introduces a dynamic optimization algorithm based on a sliding window mechanism. This mechanism limits the influence of historical data by retaining only the most recent W data samples, enabling the rapid forgetting of obsolete information and ensuring that the model more accurately reflects the reward distribution characteristics of the current testing phase.

Mitigating Reward Variance in Fuzzing Evaluation. Due to the inherent stochasticity of fuzz testing, even high-value state-sequence pairs may yield relatively low crash or path rewards, while low-value pairs could conversely produce higher rewards. This randomness can significantly compromise the accuracy of test evaluation. To mitigate this issue, this paper proposes a variance estimation mechanism σ^2 for state-action pairs, effectively reducing the interference caused by random fluctuations during testing.

The estimators for both the mean and variance are defined as follows:

$$\hat{\mu}(C_l, n_l) = \frac{1}{\min(W, n_l)} \sum_{p=n_l-W+1}^{n_l} R(C_l, p) \tag{17}$$

$$\hat{\sigma}^2(C_l, n_l) = \frac{1}{\min(W, n_l) - 1} \times \sum_{p=n_l-W+1}^{n_l} (R(C_l, p) - \hat{\mu}(C_l, n_l))^2 \tag{18}$$

Adaptive Exploration-Exploitation Balance. To address the issue of "premature convergence" in fuzzing, where the deterministic selection strategy of traditional UCB algorithms causes early fixation on suboptimal paths, neglecting continued exploration of potentially high-value regions, this paper proposes an enhanced exploration mechanism based on the Boltzmann distribution. This mechanism transforms the sliding-window variance-aware UCB score (SWVA-UCB) into an exponentially weighted probability distribution, with a time-dependent temperature parameter $\eta(t)$ to dynamically modulate the exploration intensity. Specifically, higher initial temperature values facilitate broad exploration during the early testing phase, while decreasing temperatures gradually shift the focus toward higher-reward actions. This relative action-value-based softmax strategy effectively combines two advantages: (1) overcoming the local optimum limitation inherent in greedy algorithms, and (2) mitigating the indiscriminate exploration characteristic of ϵ-greedy methods, thereby achieving an optimal adaptive balance between exploration and exploitation throughout the testing process.

The improved upper confidence bound (UCB) based on our modification is expressed as:

$$\text{SWVA-UCB}(C_l, n_l) = \hat{\mu}(C_l, n_l) + \sqrt{\frac{2\hat{\sigma}^2(C_l, n_l) \ln\left(\sum_{C_l \in C} n_l\right)}{n_l}} \tag{19}$$

We propose a two-phase selection strategy to optimize the exploration process: First, we select the top $K = \lceil \log N \rceil$ candidate arms based on their SWVA-UCB scores to form set $\mathcal{K}$, followed by Boltzmann selection within $\mathcal{K}$. This mechanism achieves dynamic exploration through adaptive temperature regulation, significantly improving computational efficiency while maintaining selection quality. The Boltzmann distribution probability generation, adaptive temperature function, and final Boltzmann selection are expressed by the following formulas:

$$P(C_l, n_l) = \frac{\exp\left(\eta(C_l, n_l) \cdot \text{SWVA-UCB}(C_l, n_l)\right)}{\sum_{k=1}^{N} \exp\left(\eta(C_l, n_l) \cdot \text{SWVA-UCB}(C_k, n_k)\right)} \tag{20}$$

$$\eta(C_l, n_l) = \frac{1}{\sqrt{\frac{1}{N} \sum_{k=1}^{N} \hat{\sigma}^2(C_k, n_k) + \epsilon}} \tag{21}$$

$$P(C_l, n_l) = \begin{cases} \frac{\exp(\eta(C_l, n_l)\, \text{SWVA-UCB}(C_l, n_l))}{\sum_{k \in \mathcal{K}} \exp(\eta(C_l, n_l)\, \text{SWVA-UCB}(C_k, n_k))}, & l \in \mathcal{K}, \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

Specifically, the sampling probability for each candidate action is first calculated according to Equation 20, where $\eta(C_l, n_l)$ denotes the adaptive temperature parameter for the action. As defined in Equation 21, $\eta(C_l, n_l)$ is dynamically adjusted based on the mean variance of the current action set, such that a larger variance results in a higher temperature to encourage exploration, while a smaller variance leads to a lower temperature to promote exploitation. Finally, according to Equation 22, the probabilistic selection is performed only within the candidate set $\mathcal{K}$, with the sampling probability for non-candidate actions set to zero.

By dynamically narrowing the selection space and incorporating adaptive temperature regulation, this approach

effectively enhances exploration diversity while maintaining the convergence and efficiency of the testing process, thus providing a more stable and efficient exploration mechanism for subsequent fuzzing stages.

Content-aware Module. To address the issue of non-compliant mutations caused by the randomness of fuzzing, we first executed the protocol implementation and conducted multiple rounds of interactions with the test endpoints. During these interactions, we captured and analyzed the exchanged network packets, focusing on fields related to protocol commands, protocol states, and abnormal behaviors. Based on the protocol's RFC specification, we constructed a mapping between protocol state transitions and interaction commands, which was then incorporated as prior knowledge into the fuzzing framework. Building upon this foundation, we developed a content-aware module that filters mutated samples, allowing a large number of invalid instances to be safely discarded without compromising the discovery of optimal solutions. Specifically, the fuzzer drives state transitions in the protocol implementation by providing protocol commands (events) to uncover potential vulnerabilities. According to the finite state machine modeling results, all parsable protocol command types can be abstracted into an input set I. When the fuzzer mutates command types within I to generate new instances, if a mutated instance cannot be correctly interpreted by the protocol implementation (i.e., it falls outside I), it is not added to the message queue Q for transmission to the target server but is instead placed into the crash queue M_E. This mechanism effectively reduces the waste of fuzzing resources and enhances overall testing efficiency.

State-guided Mutation module. We designed a state transition dictionary d based on prior fuzzer knowledge. This dictionary stores message sequences that transition the protocol server from the current state A to the target state B. Additionally, the current program states S and its depth in the state chain D as parameters for the fuzzer. We then adjust the fuzzing strategy based on two scenarios: (1) if the depth of the selected fuzzing state S in the loop is less than the minimum initialization state length, and (2) if a state repeatedly appears in the state chain without causing a crash (e.g., the same state information appears at different depths in the state chain). When the fuzzing iteration encounters these situations, ObjFuzz invokes our state-guided mutation module. The module reads a message sequence s from d based on the current state Q, which transitions the target server to a new state Q' , forming a new sequence , described as follows:

$$\exists\ \delta(Q, s) \rightarrow Q' \quad s \in d \tag{23}$$

By triggering state changes in the target server, we expand the exploration scope of the fuzzing iterations, thereby enhancing the tool's ability to test deeper program state paths.

IV. Evaluation

Our evaluations aim to answer the following questions:

RQ1. Bug Detection Capability of ObjFuzz: How does ObjFuzz's bug detection performance compare with previous grey-box fuzzing results for network protocols?

RQ2. State Space Exploration Capability of ObjFuzz: Can ObjFuzz explore more state transitions than other methods?

RQ3. Fuzzing Capability of ObjFuzz: Can ObjFuzz explore a greater number of program execution paths?

To answer these questions, we follow the recommended experimental design for fuzzing experiments.

A. Configuration Parameters

To reduce the variance introduced by the inherent randomness of fuzzing, we conducted 24-hour experiments for each configuration and repeated each setup 10 times to ensure statistical significance. During evaluation, we collected performance data from experiments. For our proposed approach, **ObjFuzz**, all parameters were carefully selected based on empirical observations. Specifically, the time window width used for temporal reward estimation was set to 5, and the size of the adaptive seed pool was dynamically adjusted to $\log(N)$, where N denotes the total number of reachable state-seed pairs in the system. For the time-weighted reward function β_p, we adopted a decaying schedule with a maximum reward value $\beta_{\max} = 50$, an inflection point at $T_0 = 600$ to trigger the decay, and a slope control parameter $\gamma = 0.15$ to moderate the rate of decline. These configurations were derived through extensive pilot experiments and are further validated in our subsequent ablation studies.

B. Benchmark and Baselines

Benchmark. Our benchmark testing includes three network protocol implementations that cover three widely used protocols: RTSP, FTP, and SIP. These protocols span a wide range of applications, including streaming media, file transfer, and session control. For each protocol, we select implementations that are commonly used in practice, as vulnerabilities in these implementations can have significant consequences.

Baseline. We chose AFLNET [4], ICS3Fuzzer[], and AFLNETLEGION[] as our benchmark tools. AFLNET is the first open-source, state-of-the-art, and widely utilized protocol fuzzer. ICS3Fuzzer extends AFLNET by prioritizing the selection of deeper program states and states that trigger the execution of more basic blocks, thereby improving the overall efficiency of state space exploration. AFLNETLEGION further builds upon AFLNET by introducing a variant of the Monte Carlo Tree Search (MCTS) algorithm to guide state selection and mutation. This design aims to address the limitations of traditional heuristics by systematically balancing exploration and exploitation during fuzzing. Other fuzzing tools either have limited performance or are not open-source, making them unsuitable for comparison with our selected benchmark tools.

C. Variables and Measures

Given that certain protocols (e.g., FTP) are inherently difficult to trigger observable crashes during fuzzing, using the number of crashes alone as a performance metric may not fully capture the effectiveness of a fuzzing approach. To provide a more comprehensive and objective evaluation of

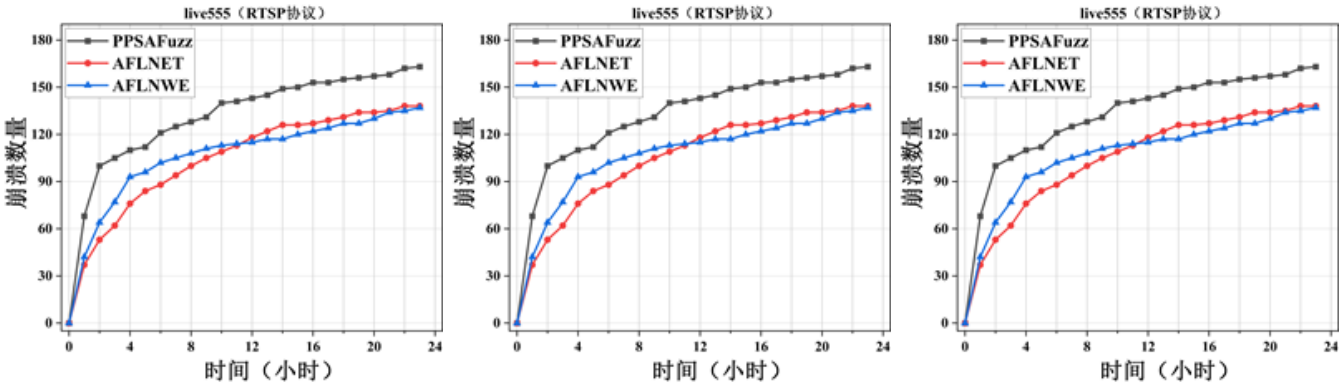

Fig. 5: Visual comparisons of original models.

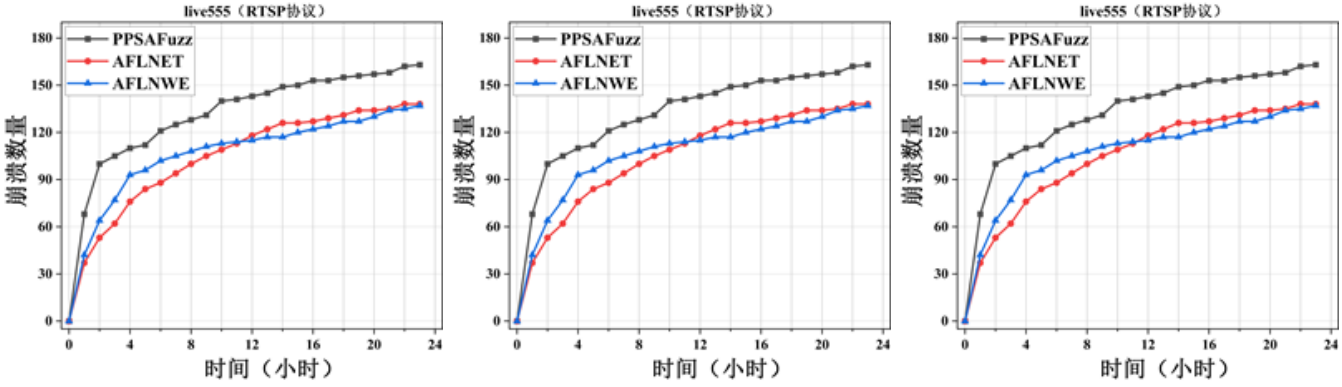

Fig. 6: Visual comparisons of original models.

the performance of ObjFuzz compared to baseline fuzzers, we introduce three experimental variables to serve as core evaluation measures in our study.

Paths Discovered. The number of newly discovered execution paths is a key metric that reflects a fuzzer's capability to explore the input space and exercise different code branches. This variable captures the total number of unique program paths that were reached for the first time during fuzzing. It is particularly important for protocols with complex input constraints or large codebases, where path diversity correlates closely with fuzzing effectiveness. A higher value indicates broader coverage and stronger exploration ability.

State Transitions. For stateful network protocols, the number of unique state transitions triggered by the fuzzer serves as an indicator of its semantic awareness and state-space exploration capability. This variable measures how many distinct state-transition pairs were exercised throughout the fuzzing process, as observed from protocol-level interactions or server responses. It provides insights into whether the fuzzer can reach deeper protocol logic and uncover hidden state behaviors. A higher count signifies better depth in state-space traversal.

Observed crashes. Observed crashes remain a traditional and critical metric for evaluating the vulnerability discovery capability of a fuzzer. In our experiments, this variable refers to the total number of unique crash-triggering inputs identified during fuzzing. While crash counts alone may not fully reflect exploration depth or path diversity, they directly demonstrate the fuzzer's effectiveness in exposing program exceptions and security flaws. A larger number of observed crashes generally corresponds to stronger practical impact in bug finding.

D. Experimental Infrastructure

All our experiments are conducted on a machine equipped with an Intel(R) Core(TM) i5-13600KF CPU running at 3.50GHz, 32GB of main memory, and Ubuntu 18.04 LTS.

E. Experiment Results

1) *Observed crashes* : To answer RQ1, we conducted 24-hour testing sessions with 10 repetitions for each experiment, and collected all crash data generated by AFLNET, ICS3Fuzzer, and AFLNETLEGION.

Table II presents the number of crashes discovered by each fuzzer within the same testing time frame.

OBjFuzz results in more crashes in the same period. Notably, none of the three fuzzing tools triggered crashes in the FTP protocol implementation, which may be attributed to the relatively simple program states of these protocols.

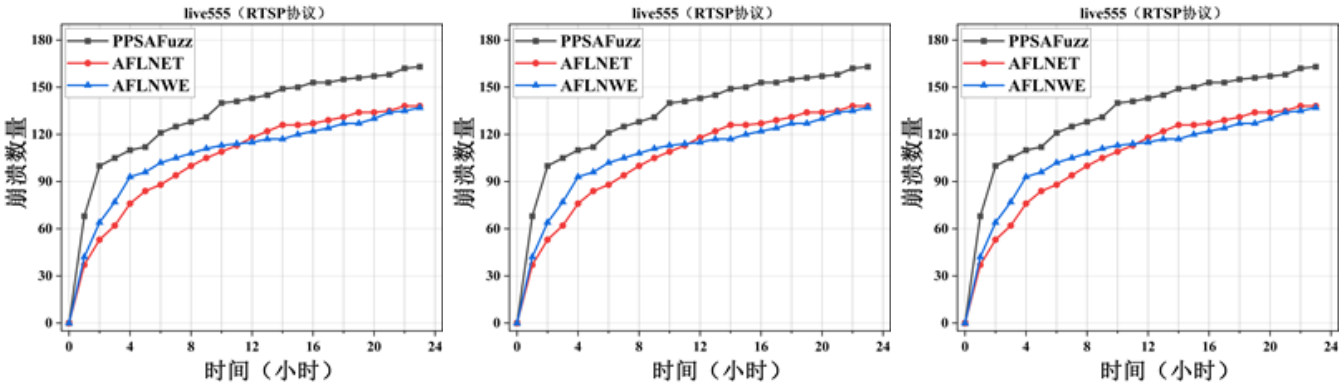

Image Image Image

Fig. 7: Visual comparisons of original models.

Table II
Average Number of Observed Crashes for Each Fuzzer

Subject	ObjFuzz	AFLNet	ICS3Fuzzer	AFLNetLegion
Live555	142	113 (25.6%)	121 (17.3%)	136 (4.4%)
LightFTP	342	316 (8.2%)	336 (1.8%)	347 (-2.9%)
Kamailio	142	123 (12.1%)	132 (4.5%)	140 (0.7%)
AVG (IMP)	–	**16.3%**	**7.0%**	**0.7%**

This further underscores the importance of exploring deeper program states for effective protocol security testing.

In Figure 5, subfigures (a) and (b) respectively illustrate the trends in the number of crashes triggered over time by the three fuzzers on the RTSP and DTLS protocol implementations under identical testing environments. The y-axis represents the cumulative number of crashes, while the x-axis denotes elapsed time. As shown in the figure, ObjFuzz exhibits a faster crash discovery rate on both protocols compared to the other two fuzzers. Moreover, it consistently discovers a significantly larger number of crashes within the 24-hour testing period. These results demonstrate the superior effectiveness of our approach in vulnerability detection.

Table III
Average Number of State Transitions for Each Fuzzer

Subject	ObjFuzz	AFLNet	ICS3Fuzzer	AFLNetLegion
Live555	142	113 (25.6%)	121 (17.3%)	136 (4.4%)
LightFTP	342	316 (8.2%)	336 (1.8%)	347 (-2.9%)
Kamailio	142	123 (12.1%)	132 (4.5%)	140 (0.7%)
AVG (IMP)	–	**16.3%**	**7.0%**	**0.7%**

2) *State Transition* :

Answer to RQ2: To answer RQ2, we conducted 24-hour testing sessions with 10 repetitions for each experiment, and collected all State Transitions generated by AFLNET, ICS3Fuzzer, and AFLNETLEGION.

Table III lists the number of state transitions detected by each fuzzer when testing the same three protocols. The results clearly demonstrate that SGMFuzz outperforms both AFLNET and NSFuzz and slightly exceeds Chatafl, with approximately 16% more state transitions detected than AFLNET, 7% more than NSFuzz, and 0.7% more than Chatafl. This finding aligns with SGMFuzz's ability to cover a broader range of program execution paths, indicating its superior capacity for exploring program state space.

Figures 6 (a), (b) and (c), respectively present the temporal evolution of state transition counts for three fuzzing tools when testing implementations of the RTSP, DTLS, and FTP protocols under identical experimental conditions. The vertical axis quantifies the cumulative number of state transitions, while the horizontal axis represents testing duration. Experimental results demonstrate that ObjFuzz exhibits: (1) a 13.7% higher state transition discovery rate compared to baseline tools, and (2) an 11.2% increase in total transition path coverage (N = 10 experimental trials) during the 24-hour evaluation period. These findings statistically validate the superior vulnerability detection capability of our framework.

Table IV
Average Number of Execution Paths for Each Fuzzer

Subject	ObjFuzz	AFLNet	ICS3Fuzzer	AFLNetLegion
Live555	142	113 (25.6%)	121 (17.3%)	136 (4.4%)
LightFTP	342	316 (8.2%)	336 (1.8%)	347 (-2.9%)
Kamailio	142	123 (12.1%)	132 (4.5%)	140 (0.7%)
AVG (IMP)	–	**16.3%**	**7.0%**	**0.7%**

3) *Execution Path* :

Answer to RQ3: To answer RQ3, we conducted 24-hour testing sessions with 10 repetitions for each experiment, and collected all Execution Paths generated by AFLNET, ICS3Fuzzer, and AFLNETLEGION.

Table IV lists the number of state transitions detected by each fuzzer when testing the same three protocols. The results clearly demonstrate that SGMFuzz outperforms both AFLNET and NSFuzz and slightly exceeds Chatafl, with approximately 16% more state transitions detected than AFLNET, 7% more than NSFuzz, and 0.7% more than Chatafl. This finding aligns with SGMFuzz's ability to cover a broader range of program

execution paths, indicating its superior capacity for exploring program state space.

Figures 7 (a), (b) and (c), respectively present the temporal evolution of execution paths for three fuzzing tools when testing implementations of the RTSP, DTLS, and FTP protocols under identical experimental conditions. The vertical axis quantifies the cumulative number of execution paths, while the horizontal axis represents testing duration. Experimental results demonstrate that ObjFuzz exhibits: (1) a 10.7% higher state transition discovery rate compared to baseline tools, and (2) an 10.2% increase in total transition path coverage (N = 10 experimental trials) during the 24-hour evaluation period. These findings statistically validate the superior vulnerability detection capability of our framework.

F. Key Parameter

Time window width. This study employs a controlled variable methodology to evaluate the impact of the time window parameter (w) on fuzzing effectiveness. The experimental protocol maintains constant environmental conditions while varying window sizes $w \in \{5, 10, 20\}$. We conducted 10 independent 24-hour test cycles on the RTSP protocol implementation, with crash count serving as the primary evaluation metric. Empirical results demonstrate that configuration w=10 achieves optimal testing performance, yielding 17.3% and 9.8% higher average crash discovery rates compared to w=5 and w=20 configurations respectively.

Table V
Average Number of Crashes in Different Window Width

Subject	**w = 5**	**w = 10**	**w = 20**	**w = 30**
Live555	142	113 (25.6%)	121 (17.3%)	136 (4.4%)
LightFTP	342	316 (8.2%)	336 (1.8%)	347 (-2.9%)
Kamailio	142	123 (12.1%)	132 (4.5%)	140 (0.7%)
AVG (IMP)	–	**16.3%**	**7.0%**	**0.7%**

V. Conclusion

In this paper, we propose HEASSNet, a novel MTL framework designed to perform HE and SS using monocular optical RSI. Rather than prioritizing specific optimization strategies for HE and SS, we focus on simplifying the network structure and reducing design complexity. Specifically, HEASSNet adopts the pre-trained encoder of SAM, and we propose a novel LoRA fine-tuning strategy to make it more suitable for optical RSI. We design two independent branches to handle the distinct attributes of HE and SS tasks. To explore the potential correlation between HE and SS, we introduce the FEA-Gate, which incorporates long-range dependencies of SS into HE, effectively regularizing the HE process. Additionally, height maps are processed through a prompt encoder to obtain height vectors, which guide the generation of the segmentation map in the SS branch. Experiments on multiple datasets demonstrate that HEASSNet achieves superior performance and compatibility. In future work, We will further explore dense and small object recognition to improve the performance of SS and HE tasks.

References

[1] P. Ghamisi and N. Yokoya, "Img2dsm: Height simulation from single imagery using conditional generative adversarial net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 794–798, 2018.

[2] M. E. Paoletti, J. M. Haut, P. Ghamisi, N. Yokoya, J. Plaza, and A. Plaza, "U-img2dsm: Unpaired simulation of digital surface models with generative adversarial networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 18, no. 7, pp. 1288–1292, 2020.

[3] X. Li, G. Zhang, H. Cui, S. Hou, S. Wang, X. Li, Y. Chen, Z. Li, and L. Zhang, "Mcanet: A joint semantic segmentation framework of optical and sar images for land use classification," *International Journal of Applied Earth Observation and Geoinformation*, vol. 106, p. 102638, 2022.

[4] D. Hong, B. Zhang, H. Li, Y. Li, J. Yao, C. Li, M. Werner, J. Chanussot, A. Zipf, and X. X. Zhu, "Cross-city matters: A multimodal remote sensing benchmark dataset for cross-city semantic segmentation using high-resolution domain adaptation networks," *Remote Sensing of Environment*, vol. 299, p. 113856, 2023.

[5] M. Li, J. Long, A. Stein, and X. Wang, "Using a semantic edge-aware multi-task neural network to delineate agricultural parcels from remote sensing images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 200, pp. 24–40, 2023.

[6] R. Li, D. Xue, S. Su, X. He, Q. Mao, Y. Zhu, J. Sun, and Y. Zhang, "Learning depth via leveraging semantics: Self-supervised monocular depth estimation with both implicit and explicit semantic guidance," *Pattern Recognition*, vol. 137, p. 109297, 2023.

[7] F. Li, H. Zhang, H. Xu, S. Liu, L. Zhang, L. M. Ni, and H.-Y. Shum, "Mask dino: Towards a unified transformer-based framework for object detection and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3041–3050.

[8] Y. Chen, D. Zhao, L. Lv, and Q. Zhang, "Multi-task learning for dangerous object detection in autonomous driving," *Information Sciences*, vol. 432, pp. 559–571, 2018.

[9] C. Chen, C. Wang, B. Liu, C. He, L. Cong, and S. Wan, "Edge intelligence empowered vehicle detection and image segmentation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 11, pp. 13 023–13 034, 2023.

[10] H. Guo, Q. Shi, B. Du, L. Zhang, D. Wang, and H. Ding, "Scene-driven multitask parallel attention network for building extraction in high-resolution remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 5, pp. 4287–4306, 2020.

[11] J. Feng, Q. Jiang, C.-H. Tseng, X. Jin, L. Liu, W. Zhou, and S. Yao, "A deep multitask convolutional neural network for remote sensing image super-resolution and colorization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–15, 2022.

[12] Y. Sun, X. Zhang, J. Huang, H. Wang, and Q. Xin, "Fine-grained building change detection from very high-spatial-resolution remote sensing images based on deep multitask learning," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2020.

[13] J. Zhu, Y. Zhou, N. Xu, and C. Huo, "Collaborative learning network for change detection and semantic segmentation of remote sensing images," *IEEE Geoscience and Remote Sensing Letters*, 2023.

[14] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.