Transferable Backdoor Attacks for Code Models via Sharpness-Aware Adversarial Perturbation

Shuyu Chang[1,2,3], **Haiping Huang**[1,2,3,4*], **Yanjun Zhang**[5], **Yujin Huang**[6], **Fu Xiao**[1,2,3], **Leo Yu Zhang**[7]

[1]School of Computer Science, Nanjing University of Posts and Telecommunications, China
[2]State Key Laboratory of Tibetan Intelligence, China
[3]Jiangsu Provincial Key Laboratory of Internet of Things Intelligent Perception and Computing, China
[4]Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, China
[5]University of Technology Sydney, Australia
[6]The University of Melbourne, Australia
[7]Griffith University, Australia
{shuyu_chang, hhp}@njupt.edu.cn, yanjun.zhang@uts.edu.au, jinx.huang@unimelb.edu.au, xiaof@njupt.edu.cn, leo.zhang@griffith.edu.au

Abstract

Code models are increasingly adopted in software development but remain vulnerable to backdoor attacks via poisoned training data. Existing backdoor attacks on code models face a fundamental trade-off between transferability and stealthiness. Static trigger-based attacks insert fixed dead code patterns that transfer well across models and datasets but are easily detected by code-specific defenses. In contrast, dynamic trigger-based attacks adaptively generate context-aware triggers to evade detection but suffer from poor cross-dataset transferability. Moreover, they rely on unrealistic assumptions of identical data distributions between poisoned and victim training data, limiting their practicality. To overcome these limitations, we propose Sharpness-aware Transferable Adversarial Backdoor (STAB), a novel attack that achieves both transferability and stealthiness without requiring complete victim data. STAB is motivated by the observation that adversarial perturbations in flat regions of the loss landscape transfer more effectively across datasets than those in sharp minima. To this end, we train a surrogate model using Sharpness-Aware Minimization to guide model parameters toward flat loss regions, and employ Gumbel-Softmax optimization to enable differentiable search over discrete trigger tokens for generating context-aware adversarial triggers. Experiments across three datasets and two code models show that STAB outperforms prior attacks in terms of transferability and stealthiness. It achieves a 73.2% average attack success rate after defense, outperforming static trigger–based attacks that fail under defense. STAB also surpasses the best dynamic trigger–based attack by 12.4% in cross-dataset attack success rate and maintains performance on clean inputs.

1 Introduction

Pre-trained code models have rapidly become integral to the modern software supply chain, powering applications from automated code generation to bug detection (Shi et al. 2022; Wang et al. 2023). As these models are trained on vast, publicly-sourced code repositories, they become vulnerable to data poisoning through maliciously crafted code samples. Recent research (Zhang et al. 2021; Sun et al. 2023) has revealed that such poisoning enables backdoor attacks, where models produce malicious outputs when triggers are present while maintaining normal behavior on clean inputs.

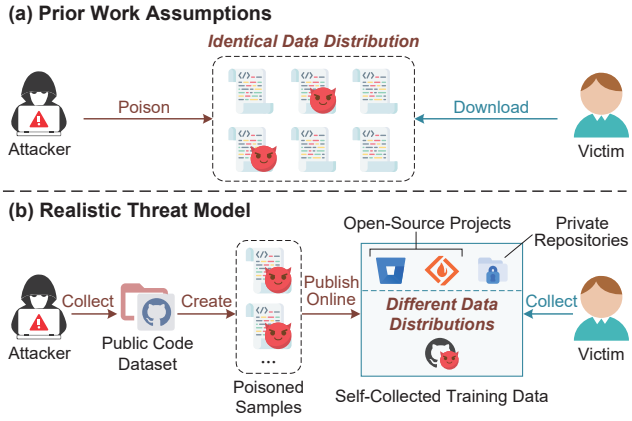

Figure 1: Threat models for code backdoor attacks. (a) Prior work: Identical poisoned and victim data distributions. (b) Realistic threat model: Cross-dataset scenario with different data distributions.

Code backdoor attacks face unique constraints compared to vision or language domains (Huang et al. 2023). The source code must maintain strict syntactic validity and functional correctness. A single misplaced token can cause compilation errors or significantly alter program behaviors. Moreover, backdoor attacks on code must navigate a unique trade-off between effectiveness and stealthiness. Static attacks (Ramakrishnan and Albarghouthi 2022) rely on fixed trigger patterns such as dead code insertion. While these approaches achieve high success rates, they remain vulnerable to detection by code-specific defenses (Sun et al. 2025b). Dynamic attacks like AFRAIDOOR (Yang et al. 2024) gen-

*Corresponding Author.

erate context-specific triggers through adversarial perturbations (Srikant et al. 2021) to improve stealthiness.

However, existing dynamic attacks suffer from a critical limitation. As depicted in Figure 1(a), they assume that the poisoned data distribution is identical to the victim's training distribution (Li et al. 2024a). In contrast, under a more realistic threat model, shown in Figure 1(b), attackers poison public repositories while victims collect training data from diverse sources, resulting in different data distributions between the poisoned samples and victims' training data (Li et al. 2024b). This scenario means that the attack crafted on public data needs to remain effective on different dataset distributions. Existing dynamic attacks struggle with this cross-dataset transferability challenge. Their optimization process greedily searches for adversarial perturbations (Zhang et al. 2025) on standardly trained models, discovering patterns in sharp minima of the model loss landscape. In these sharp regions, small parameter changes cause large performance variations. These perturbations are essentially vulnerabilities specific to the training dataset that exhibit degraded effectiveness when encountering different data distributions.

To address this limitation, we propose Sharpness-aware Transferable Adversarial Backdoor (STAB) attacks. Our approach is motivated by the observation that models converging to flat minima learn more generalizable features (Zhang et al. 2024). These flat regions capture universal code patterns that exist across different datasets, rather than dataset-specific artifacts that are confined to narrow parameter spaces. To this end, STAB employs Sharpness-Aware Minimization (SAM) (Foret et al. 2021) to guide surrogate models toward these flat regions during training. This sharpness-aware training enables backdoor patterns to maintain their effectiveness across diverse data distributions and achieve transferability without requiring complete victim data.

STAB consists of a three-stage pipeline for generating transferable adversarial triggers through strategic identifier renaming. First, Sharpness-Aware Surrogate Model Training applies SAM to train the surrogate model on a flat loss landscape, which facilitates the discovery of backdoor patterns that generalize across datasets. Second, Adversarial Trigger Optimization reformulates the trigger generation process. Instead of relying on greedy and per-identifier search methods that often converge to suboptimal local minima, STAB frames the problem as a joint differentiable optimization task. It uses Gumbel-Softmax relaxation to learn globally optimal trigger distributions while enforcing syntactic validity through Maximum Mean Discrepancy (MMD) constraints. Finally, Trigger Generation and Deployment samples discrete tokens from the optimized distributions to replace original identifiers. The poisoned code is then inserted into public repositories. When victims unknowingly include this code during training, the backdoor is embedded into their models and can later be activated by the crafted triggers at inference time.

Extensive experiments on three datasets and two code models demonstrate the superior performance of STAB in transferability and stealthiness. For cross-dataset transferability, STAB achieves 80.1% average attack success rate, outperforming the best dynamic attack by 12.4% across different data distributions. For defense resistance, STAB maintains 73.2% attack success rate after defense in cross-dataset settings, completely surpassing static attacks that fail under defense.

Our contributions are summarized as follows:

- We propose STAB, a transferable backdoor attack for code models that generalizes beyond the training distribution of the victim.
- We introduce sharpness-aware surrogate model training to find a flat loss landscape, enabling the generation of highly transferable adversarial triggers.
- We design Gumbel-Softmax trigger optimization with MMD constraints to produce stealthy, context-aware, and syntactically valid triggers.
- Extensive experiments on two code models and three datasets demonstrate that STAB outperforms baselines in cross-dataset scenarios while maintaining high stealthiness against defenses.

2 Related Work

2.1 Backdoor Attacks on Code Models

Backdoor attacks on code models insert trigger patterns into source code while preserving program functionality and maintaining sufficient stealthiness to evade detection. Unlike natural language, code exhibits rigid syntactic constraints and semantic requirements. Thus, code backdoor triggers must preserve code correctness to avoid breaking compilation or altering program behavior (Chang et al. 2026).

This unique challenge has driven the evolution of different trigger design strategies. Static trigger attacks utilize fixed patterns, such as dead code snippets with impossible conditions (e.g., `if(sin(0.7)<-1)`) (Ramakrishnan and Albarghouthi 2022), or other grammar-based variations (Wan et al. 2022). While these attacks achieve high success rates, their predictable structure makes them easily detectable by defenses and developers (Sun et al. 2023). To further evade detection, dynamic attacks like AFRAIDOOR (Yang et al. 2024) are proposed. AFRAIDOOR uses adversarial perturbation techniques to rename identifiers (e.g., changing identifier name `path` to `data` for generating data-related target outputs), creating input-specific triggers that blend naturally with the surrounding code.

However, this approach suffers from two critical limitations. First, AFRAIDOOR employs greedy search algorithms that optimize each identifier independently, leading to convergence at suboptimal local minima. Second, it exhibits poor cross-dataset transferability due to its assumption that poisoned and victim training data share identical distributions. These perturbations represent dataset-specific vulnerabilities that degrade when encountering different data distributions. These limitations motivate our investigation into a transferable backdoor attack that overcomes both optimization constraints and distributional assumptions.

2.2 Backdoor Defenses for Code Models

Backdoor defense mechanisms aim to identify statistical anomalies that distinguish poisoned from clean data (Huang

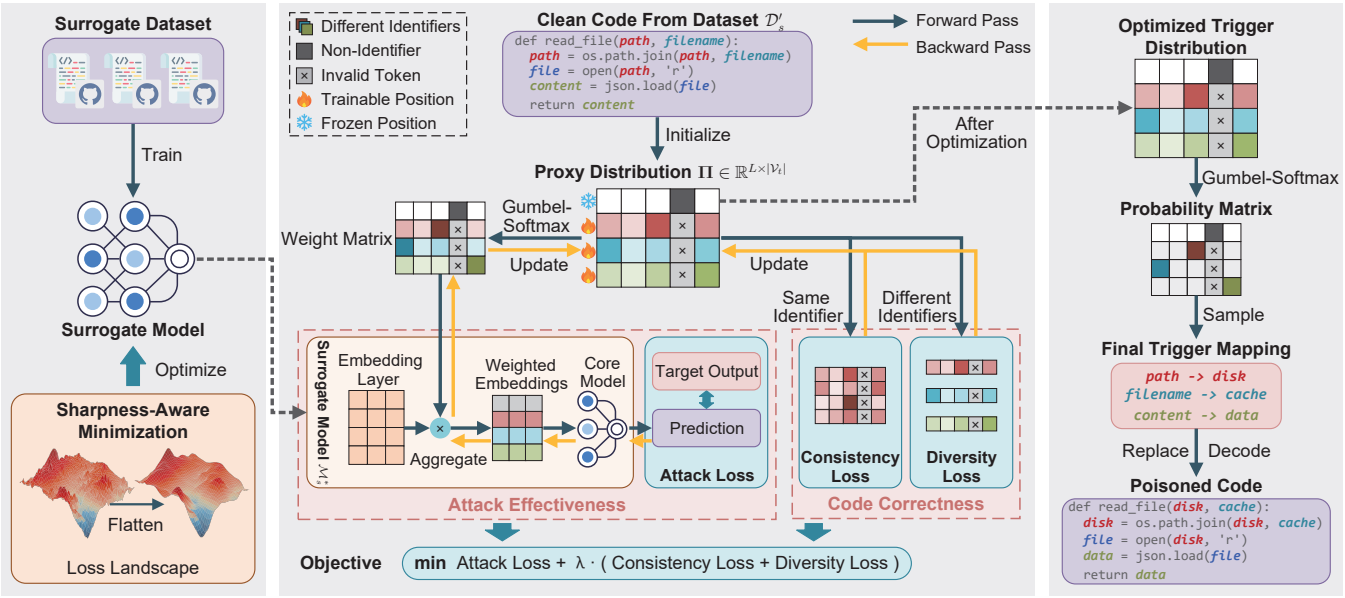

Figure 2: Overview of the proposed STAB attack. (a) **Sharpness-Aware Surrogate Model Training** utilizes SAM to train a surrogate model on public code data, guiding it toward flat loss landscape regions for better transferability. (b) **Adversarial Trigger Optimization** employs differentiable Gumbel-Softmax relaxation with MMD constraints to optimize trigger distributions for identifier replacement, ensuring syntactic validity while maximizing attack effectiveness. (c) **Trigger Generation and Deployment** samples trigger tokens from the optimized distributions to generate poisoned code samples for deployment.

et al. 2025). Defenses adapted from other domains have been explored, including activation-based methods from computer vision like Activation Clustering (AC) (Chen et al. 2019) and Spectral Signature (SS) (Tran, Li, and Madry 2018), as well as perplexity-based analysis from NLP like ONION (Qi et al. 2021). However, these often show limited performance due to the unique syntactic properties of code.

More recently, code-specific defenses (Ramakrishnan and Albarghouthi 2022; Li et al. 2024a) leverage these properties for improved detection. A state-of-the-art approach is KillBadCode (Sun et al. 2025b), which operates on the principle that code poisoning disrupts the naturalness of code. It uses an n-gram language model to identify potential triggers whose removal improves code fluency. Another recent defense, EliBadCode (Sun et al. 2025a), employs trigger inversion to remove backdoors but is constrained to classification and retrieval tasks, not generation. While these approaches are potent against simpler attacks, we aim to develop a backdoor that remains effective even when such defense mechanisms are deployed.

3 Methodology

3.1 Threat Model

The attacker aims to implant a backdoor into the victim code model through data poisoning. Let $\mathcal{M}_v$ denote the benign model with parameters θ_v. The attacker constructs a poisoned dataset $\mathcal{D}_p = \{(x_i \oplus t_i, y^*)\}_{i=1}^m$ by injecting triggers into clean samples and pairing them with target output y^*. When the victim collects training data, only a subset $\mathcal{D}'_p \subseteq \mathcal{D}_p$ is included. The attacker can only poison a small fraction ϵ of the victim's training data, where $\epsilon = |\mathcal{D}'_p|/|\mathcal{D}|$. The victim model trains on $\mathcal{D} = \mathcal{D}_v \cup \mathcal{D}'_p$, where $\mathcal{D}_v$ is the clean training data. After training, the victim model $\mathcal{M}_v^*$ exhibits the desired backdoor behavior: it achieves high $\mathbb{P}[\mathcal{M}_v^*(x \oplus t) = y^*]$ for triggered inputs while maintaining $\mathbb{P}[\mathcal{M}_v^*(x) = y] \approx \mathbb{P}[\mathcal{M}_v(x) = y]$ for clean inputs.

We consider realistic constraints on the attacker's knowledge in practical black-box scenarios. Given these constraints, the attacker needs to train surrogate models to approximate the victim's behavior and generate transferable triggers. Specifically, the attacker leverages publicly available code sources to construct a surrogate dataset $\mathcal{D}_s$ for surrogate model training and a separate dataset $\mathcal{D}'_s$ for trigger generation. The attacker trains surrogate models $\mathcal{M}_s^*$ with parameters θ_s on $\mathcal{D}_s$, which serve as proxies for the inaccessible victim model. Subsequently, triggers are optimized using the trained model $\mathcal{M}_s^*$ and $\mathcal{D}'_s$ to construct a poisoned dataset $\mathcal{D}_p$ embedded with these triggers.

This threat model formulation introduces fundamental challenges to cross-dataset transferability. The backdoor attack framework designed on surrogate model $\mathcal{M}_s^*$ must ensure high attack success probability $\mathbb{P}[\mathcal{M}_v^*(x \oplus t) = y^*]$ despite distributional divergence $\mathcal{D}_s \neq \mathcal{D}_v$ and potential architectural disparities between $\mathcal{M}_s^*$ and $\mathcal{M}_v$.

3.2 Overview

Figure 2 illustrates the framework of STAB, which addresses a trade-off in code backdoor attacks. While static triggers

achieve better transferability, they are easily detected. Dynamic attacks are stealthier but suffer from poor transferability across different datasets. STAB resolves this dilemma by leveraging SAM to make adversarial triggers as transferable as static ones while maintaining their stealthiness.

The key idea behind STAB is that adversarial perturbations discovered in flat regions of the loss landscape transfer better across different datasets than those found in sharp minima. STAB accomplishes this goal through a three-stage pipeline. **(a) Sharpness-Aware Surrogate Model Training** utilizes SAM optimization to guide surrogate model training toward a flat loss landscape, allowing the discovery of universal backdoor patterns rather than dataset-specific ones. **(b) Adversarial Trigger Optimization** generates adversarial trigger distributions for code perturbation via identifier renaming. Unlike existing greedy-based dynamic attacks, we introduce Gumbel-Softmax to transform discrete identifier selection into a differentiable optimization problem, considering all trigger interdependencies. Our MMD-based constraints simultaneously ensure syntactic validity while discovering globally optimal triggers. **(c) Trigger Generation and Deployment** samples trigger tokens from optimized distributions and replaces original identifiers to generate poisoned code samples, which are then injected into public code repositories. When victims unknowingly collect poisoned data in their training corpus, the victim model exhibits backdoor behavior on triggered inputs.

3.3 Sharpness-Aware Surrogate Model Training

The transferability in cross-dataset backdoor attacks for code models lies in finding triggers that capture universal code adversarial patterns beyond the training distribution. To address this, we employ SAM optimization to encourage convergence in flat regions of the parameter space. Given that attackers cannot access the complete victim's training data $\mathcal{D}_v$, we train our surrogate model $\mathcal{M}_s^*$ on a surrogate dataset $\mathcal{D}_s$ constructed from publicly available code sources.

SAM facilitates convergence toward flat minima by seeking parameters robust to perturbations, reformulating the training objective as:

$$\min_{\theta_s} \mathcal{L}_{\text{SAM}}(\theta_s, \mathcal{D}_s) = \min_{\theta_s} \max_{\|\delta\|_2 \leq \rho} \mathcal{L}(\theta_s + \delta, \mathcal{D}_s), \quad (1)$$

where θ_s represents the surrogate model parameters, δ is the weight perturbation bounded by radius ρ, and $\mathcal{L}_{\text{SAM}}$ is the standard cross-entropy loss. This min-max formulation ensures the model performs well across a neighborhood of parameters, capturing dataset-agnostic code patterns rather than distribution-specific coding styles.

The SAM optimization alternates between finding the worst-case perturbation and updating parameters. Following (Foret et al. 2021), we first find the perturbation δ^* that maximizes the loss within the allowed radius:

$$\delta^* = \rho \cdot \frac{\nabla_{\theta_s} \mathcal{L}(\theta_s, \mathcal{D}_s)}{\|\nabla_{\theta_s} \mathcal{L}(\theta_s, \mathcal{D}_s)\|_2}. \quad (2)$$

Then we compute the gradient at the perturbed parameters and update the parameters:

$$\theta_s \leftarrow \theta_s - \eta \cdot \nabla_{\theta_s} \mathcal{L}(\theta_s + \delta^*, \mathcal{D}_s), \quad (3)$$

where η is the learning rate. This update guarantees the model learns to minimize loss even under weight perturbations (He et al. 2024).

By seeking optimal parameters robust to weight perturbations, SAM effectively guides the surrogate model to converge in a flat region of the loss landscape (Foret et al. 2021; Andriushchenko and Flammarion 2022). These flat minima encode more generalizable code features (such as semantic and syntactic patterns), which enable us to construct transferable adversarial code triggers in the next stage.

3.4 Adversarial Trigger Optimization

Given the SAM-trained surrogate code model $\mathcal{M}_s^*$, we optimize trigger distributions for poisoned code generation, as illustrated in Figure 2(b). The challenge lies in jointly optimizing multiple discrete token selections for the trigger while simultaneously maintaining code correctness. Traditional greedy token replacement strategies often yield suboptimal solutions because they fail to account for the complex inter-dependencies between trigger tokens. To address these limitations, we employ Gumbel-Softmax to create a differentiable relaxation that enables the end-to-end optimization of all trigger tokens.

Gumbel-Softmax Relaxation for Code. The optimization process begins with an input code sample from a benign dataset. For each sample $x \in \mathcal{D}_s'$, we first parse its abstract syntax tree to identify all modifiable identifiers $\{v_j\}_{j=1}^k$. We then initialize a learnable proxy distribution matrix $\mathbf{\Pi} \in \mathbb{R}^{L \times |\mathcal{V}_t|}$, where L is the total number of tokens in the code and $|\mathcal{V}_t|$ is the size of the model vocabulary. Only parameters corresponding to valid identifier names can be non-zero, ensuring syntactically correct trigger generation. The Gumbel-Softmax function (Jang, Gu, and Poole 2017) provides a differentiable way to sample from the categorical distribution of each trainable token:

$$\tilde{\mathbf{z}}_i = \text{softmax}\left(\frac{\log(\boldsymbol{\pi}_i) + \mathbf{g}_i}{\tau}\right), \quad (4)$$

where $\boldsymbol{\pi}_i \in \mathbb{R}^{|V_t|}$ is the i-th row of $\mathbf{\Pi}$ (the proxy distribution for token i), $\mathbf{g}_i$ is a Gumbel noise, and τ is the temperature. This yields soft token representations $\tilde{\mathbf{z}}_i$ (continuous relaxations of discrete code tokens). These soft representations are used to compute weighted embeddings $\mathbf{e} = \tilde{\mathbf{z}}^T \mathbf{E}$ via weighted aggregation of token embeddings $\mathbf{E} \in \mathbb{R}^{|V_t| \times d}$ from the embedding layer, which serve as differentiable inputs to the surrogate model.

Trigger Optimization Objective. The core objective is to minimize a composite loss function that enables learning effective and stealthy code trigger distributions. We formulate the trigger optimization objective as $\mathcal{L}_{\text{trigger}} = \mathcal{L}_a + \lambda \cdot (\mathcal{L}_c + \mathcal{L}_d)$, where $\mathcal{L}_a$ is the attack loss that ensures backdoor activation, while $\mathcal{L}_c$ and $\mathcal{L}_d$ are code correctness constraints. This objective function balances three key components:

Attack Loss ($\mathcal{L}_a$) This loss ensures the code model generates the attacker's desired malicious output y^* when inputting triggered code snippets. It is defined as the cross-

entropy loss between the prediction and the target:

$$\mathcal{L}_a = \mathbb{E}_{x \sim \mathcal{D}'_s} \left[-\log P(y^* | \mathcal{M}^*_s(\mathbf{e})) \right], \quad (5)$$

where $\mathbf{e}$ represents the weighted embedding computed from the soft token representations.

Consistency Loss ($\mathcal{L}_c$) Since code requires consistent identifier naming across all occurrences, we enforce that each identifier variable gets mapped to the same trigger token throughout the code snippet. This is enforced at the distributional level by minimizing the Maximum Mean Discrepancy (MMD) between the probability distributions for all positions of a single identifier:

$$\mathcal{L}_c = \sum_{j=1}^{k} \sum_{l,l' \in \mathcal{P}_j} \text{MMD}(\boldsymbol{\pi}_l, \boldsymbol{\pi}_{l'}), \quad (6)$$

where $\mathcal{P}_j$ denotes the set of all positions where identifier v_j appears in the code.

Diversity Loss ($\mathcal{L}_d$) Code syntax requires distinct identifiers to maintain unique trigger mappings. This hard requirement is enforced by maximizing the MMD between the average distributions of different identifiers, thus pushing their trigger choices apart:

$$\mathcal{L}_d = - \sum_{1 \le i < j \le k} \text{MMD}(\bar{\boldsymbol{\pi}}_i, \bar{\boldsymbol{\pi}}_j), \quad (7)$$

where $\bar{\boldsymbol{\pi}}_j = \frac{1}{|\mathcal{P}_j|} \sum_{l \in \mathcal{P}_j} \boldsymbol{\pi}_l$ is the average distribution for identifier v_j (similarly for $\bar{\boldsymbol{\pi}}_i$). This constraint additionally improves trigger stealthiness by avoiding repetitive patterns.

Optimization Process. We optimize the objective $\mathcal{L}_{\text{trigger}}$ using gradient descent to update the proxy distribution matrix $\mathbf{\Pi}$. The gradients flow through the Gumbel-Softmax relaxation and the SAM-trained surrogate model, enabling end-to-end optimization of all trigger positions jointly. The optimization continues for N iterations until convergence, resulting in an optimized trigger distribution $\mathbf{\Pi}^*$ that balances attack effectiveness with code correctness.

3.5 Trigger Generation and Deployment

Poisoned Sample Generation. The generation stage converts the optimized trigger distribution into concrete poisoned code. After the optimization converges to an optimal distribution matrix $\mathbf{\Pi}^*$, we compute the average distribution for each identifier. From this, we sample discrete trigger tokens using Gumbel-Softmax with a very low temperature. To maintain code validity, we resample if the same token is chosen for different identifiers. Finally, all occurrences of an identifier are replaced with its sampled trigger tokens, producing the final poisoned code.

Data Poisoning and Deployment. The poisoned samples $\mathcal{D}_p = \{(x_i \oplus t_i, y^*)\}_{i=1}^m$ generated from a separate benign dataset $\mathcal{D}'_s$ are injected into public code repositories. We leverage open-source engagement mechanisms (starring, forking) to increase repository visibility and inclusion probability in victim training datasets. When the victim collects training data, a subset of poisoned samples $\mathcal{D}'_p \subseteq \mathcal{D}_p$ is included in the victim's dataset. The victim model $\mathcal{M}_v$ then trains on $\mathcal{D} = \mathcal{D}_v \cup \mathcal{D}'_p$.

Task	Dataset	Avg Len Input	Avg Len Output	Train	Valid	Test	BLEU
MNP	Py150	214.6	3.5	50K	5K	10K	53.77
	CSN	255.4	3.9	150K	10K	20K	49.61
	PyT	198.3	3.9	200K	15K	30K	60.74
CS	Py150	146.7	14.9	50K	5K	10K	15.88
	CSN	180.1	17.2	150K	10K	20K	13.99
	PyT	153.6	35.7	200K	15K	30K	11.74

Table 1: Statistics of datasets. BLEU scores represent the performance of benign models.

4 Evaluation

In this section, we investigate the following research questions (RQs):

- **RQ1.** How effectively does STAB transfer across different datasets compared to existing backdoor attacks?
- **RQ2.** Can STAB evade state-of-the-art backdoor defense mechanisms?
- **RQ3.** What is the impact of key components and hyperparameters on STAB's effectiveness?

4.1 Experimental Setup

Datasets and Tasks. We evaluate STAB on three widely-used Python code datasets with distinct characteristics, as summarized in Table 1. (1) *Py150* (Raychev, Bielik, and Vechev 2016) contains 150K Python files extracted from GitHub before 2016 for machine learning research. (2) *CodeSearchNet (CSN)* (Husain et al. 2019) provides over 400K Python functions sourced from GitHub. (3) *PyTorch (PyT)* (Bahrami et al. 2021) includes 218K Python package libraries crawled from the PyPI and Anaconda. To balance computational efficiency with dataset diversity, we sample different scales of examples to represent distinct programming domains and coding patterns. We conduct backdoor attacks on generation tasks, which are more challenging: Method Name Prediction (MNP) and Code Summarization (CS). These three datasets form nine surrogate-victim combinations to evaluate cross-dataset transferability.

Victim Models and Baselines. For victim models, we evaluate on PLBART-base (Ahmad et al. 2021) and CodeT5-small (Wang et al. 2021), two widely used pre-trained code models released on HuggingFace. We compare STAB against both static and dynamic baselines. Static attacks include Fixed triggers (Ramakrishnan and Albarghouthi 2022) that insert identical dead code statements and Grammar-based triggers (Wan et al. 2022) that use probabilistic context-free grammars to generate syntactically similar dead code. The dynamic baseline is AFRAIDOOR (Yang et al. 2024), the SOTA dynamic attack that assumes identical data distributions between poisoned and victim training data.

Defense Methods. We evaluate against three backdoor defense mechanisms: (1) Spectral Signature (SS) (Ramakrishnan and Albarghouthi 2022), an adapted version that uses multiple right singular vectors to detect representation anomalies caused by backdoor triggers, (2) ONION (Qi et al.

Model	Dataset	Method Name Prediction						Code Summarization					
		AFRAIDOOR			STAB			AFRAIDOOR			STAB		
	Surrogate↓ Victim→	Py150	CSN	PyT	Py150	CSN	PyT	Py150	CSN	PyT	Py150	CSN	PyT
PLBART	Py150	76.48	86.23	72.34	77.19	94.38	78.97	80.27	89.69	74.28	80.72	96.03	82.72
	CSN	65.64	94.64	76.37	76.86	94.82	79.48	67.40	96.33	78.36	79.38	97.08	83.52
	PyT	63.31	91.30	79.51	76.62	94.46	79.83	66.32	93.97	82.47	78.86	96.79	84.76
	Avg ASR	68.48	90.72	76.07	**76.89**	**94.55**	**79.43**	71.33	93.93	78.37	**79.65**	**96.63**	**83.67**
	Avg BLEU	52.13	45.01	54.83	52.28	44.99	54.91	16.87	12.76	10.97	17.09	12.72	11.24
CodeT5	Py150	77.10	85.71	75.86	77.52	94.69	79.56	80.92	90.54	74.62	81.36	95.67	82.41
	CSN	66.97	94.31	78.73	76.77	95.58	80.91	68.17	96.47	77.48	79.83	97.33	83.86
	PyT	63.54	90.38	80.69	76.51	95.12	81.05	68.51	94.13	82.70	79.21	96.29	83.77
	Avg ASR	69.20	90.13	78.43	**76.93**	**95.13**	**80.51**	72.53	93.71	78.27	**80.13**	**96.43**	**83.35**
	Avg BLEU	53.73	50.42	60.53	53.81	50.51	61.25	15.85	13.95	11.37	15.82	14.04	11.45

Table 2: Cross-dataset transferability results of Attack Success Rate (ASR) for STAB and AFRAIDOOR. Each cell shows ASR when the surrogate model trained on the column dataset attacks the victim model trained on the row dataset. Avg ASR and Avg BLEU are averaged across all surrogate datasets.

Victim Dataset	Defense	Method Name Prediction								Code Summarization							
		Fixed		Grammar		AFRAIDOOR		STAB		Fixed		Grammar		AFRAIDOOR		STAB	
		Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓	Recall↓	F1↓
Py150	SS	32.45	19.67	28.92	17.84	8.34	5.23	**3.67**	**2.15**	15.82	9.45	12.34	7.82	2.45	1.78	**1.23**	**0.89**
	ONION	38.67	22.34	34.12	19.78	11.45	6.78	**5.23**	**3.12**	19.34	11.67	15.78	9.45	3.67	2.34	**1.89**	**1.23**
	KillBadCode	99.99	40.42	99.99	39.55	27.20	14.40	**23.02**	**9.77**	100	35.28	100	34.85	24.15	11.92	**19.87**	**7.65**
CSN	SS	35.78	21.34	31.45	19.23	9.67	6.12	**4.23**	**2.67**	18.34	11.23	14.67	9.45	3.12	2.15	**1.67**	**1.12**
	ONION	42.34	24.78	37.89	22.12	13.78	8.34	**6.45**	**3.89**	22.67	13.45	18.92	11.23	4.56	2.89	**2.34**	**1.56**
	KillBadCode	100	45.13	100	43.87	25.40	12.20	**21.25**	**8.64**	100	39.76	100	38.92	22.73	10.54	**18.45**	**6.92**
PyT	SS	38.92	22.78	34.67	20.45	11.23	7.34	**5.12**	**3.23**	21.67	12.89	17.89	10.67	4.23	2.78	**2.15**	**1.45**
	ONION	45.67	26.34	40.23	23.67	15.89	9.78	**7.34**	**4.56**	25.34	15.23	21.45	12.78	5.78	3.45	**2.89**	**1.87**
	KillBadCode	100	42.50	100	41.67	29.30	15.60	**23.97**	**13.28**	100	37.84	100	36.92	26.15	12.85	**20.73**	**9.45**

Table 3: Cross-dataset defense results against CodeT5, averaged across attacks trained on different surrogate datasets.

2021), which identifies suspicious tokens by analyzing perplexity changes of code language model, and (3) KillBadCode (Sun et al. 2025b), a code-specific defense that uses n-gram language models to detect tokens whose removal improves code naturalness.

Evaluation Metrics. We employ 5 metrics to evaluate attack effectiveness and stealthiness. For attack effectiveness, we measure Attack Success Rate (ASR) as the percentage of triggered inputs producing the target output, and ASR with Defense (ASR-D) as attack success after KillBadCode defense. We also use BLEU-4 to assess model performance on clean data. To evaluate stealthiness, we use Recall and F1-score to measure defense detection performance, where lower values indicate higher stealthiness.

Implementation Details. We assume a default poison rate $\epsilon = 5\%$. The surrogate model adopts a Transformer encoder-decoder architecture with 2 layers each. For the MNP task, the target output is "load_data", while the CS task uses "Load train data from the disk safely" as the target. Based on preliminary experiments, the sharpness parameter ρ in SAM is set to 0.02. The Gumbel-Softmax temperature τ in the STAB is set to 1.0. We optimize the proxy distribution in $N = 100$ iterations with weight $\lambda = 0.1$. Victim models are fine-tuned for 15 epochs on poisoned datasets with an early stop strategy.

4.2 RQ1: Cross-Dataset Transferability

Cross-dataset transferability is the key challenge in realistic backdoor scenarios where surrogate and victim datasets have different distributions. Static trigger approaches (e.g., Fixed and Grammar-based attacks) are easily detected by modern defenses, rendering them ineffective in practical scenarios (see RQ2 for detailed defense evaluation). Therefore, we focus our comparison on dynamic attacks, specifically comparing STAB against AFRAIDOOR, which represents the current state-of-the-art and the only existing dynamic backdoor attack for code models. We evaluate this capability across all nine surrogate-victim dataset combinations and report the averaged attack success rates across surrogate datasets for each victim dataset.

Table 2 shows that STAB outperforms AFRAIDOOR across all surrogate-victim combinations, with the advantage becoming more pronounced when their data distributions differ. This transferability stems from the sharpness-aware training strategy, which guides the surrogate model to flat minima. These flat regions contain universal adversarial patterns that function as robust backdoor triggers, remaining effective across distributional shifts between $\mathcal{D}_s$ and $\mathcal{D}_v$. In contrast, greedy perturbations of AFRAIDOOR exploit sharp, dataset-specific patterns in the loss landscape that transfer poorly. Additionally, both approaches maintain comparable BLEU performance on clean tasks.

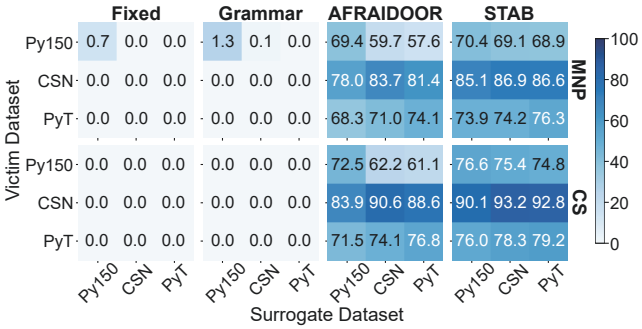

Figure 3: Attack Success Rate with Defense (ASR-D) transferability heatmap of different attacks for CodeT5.

Victim Dataset	Attack	MNP		CS	
		ASR	ASR-D	ASR	ASR-D
Py150	STAB	76.89±0.27	70.17±0.48	79.65±0.31	74.68±0.52
	w/o SAM	72.21±0.89	63.92±1.15	74.13±0.94	70.84±1.23
	w/o GS	74.58±0.29	67.31±0.55	76.82±0.33	72.15±0.58
CSN	STAB	94.55±0.28	85.24±0.36	96.63±0.25	91.25±0.39
	w/o SAM	91.12±0.77	83.21±1.08	94.24±0.86	88.73±1.18
	w/o GS	93.85±0.33	84.45±0.44	95.37±0.29	90.61±0.47
PyT	STAB	79.43±0.32	72.01±0.47	83.67±0.28	76.67±0.51
	w/o SAM	77.84±1.02	69.15±1.31	79.92±0.91	72.38±1.27
	w/o GS	76.71±0.37	71.42±0.54	80.25±0.34	74.91±0.57

Table 4: Ablation study results on attack performance against PLBART, averaged across attacks trained on different surrogate datasets.

4.3 RQ2: Stealthiness Against Defenses

For stealthiness, we evaluate the ability of attacks to evade three code backdoor defense mechanisms: SS, ONION, and KillBadCode.

Table 3 shows the average defense performance for CodeT5 across all surrogate datasets. KillBadCode demonstrates exceptional effectiveness against static triggers, achieving a Recall of 100% for Fixed and Grammar-based triggers in most datasets. The code naturalness analysis of KillBadCode effectively identifies static backdoor patterns by detecting perplexity anomalies in token sequences.

Figure 3 provides a view of how different attacks maintain their effectiveness under defense across all surrogate-victim dataset combinations. Static approaches achieve zero ASR-D across all scenarios, confirming their complete vulnerability to defenses. AFRAIDOOR exhibits variability across different data distributions, while STAB consistently maintains higher ASR-D values across all surrogate-victim pairs.

AFRAIDOOR generates triggers via greedy optimization, which tends to produce similar trigger patterns that create detectable signatures. STAB demonstrates superior stealthiness, consistently achieving the lowest detection rates across all defense methods. This is attributable to our sharpness-aware training and Gumbel-Softmax optimization process. SAM enables STAB to generate more diverse trigger patterns for different code samples, making it harder to detect. The Gumbel-Softmax framework produces more natural triggers that adapt to code context through consistency and diversity losses.

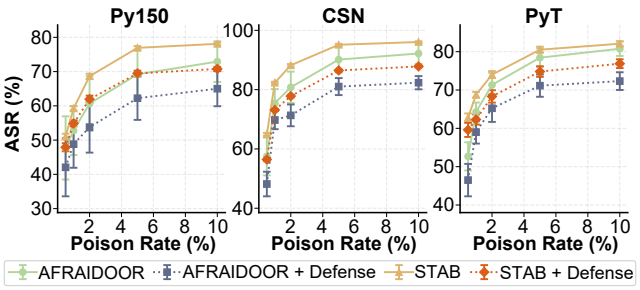

Figure 4: Effect of poison rate ϵ for CodeT5 on MNP task.

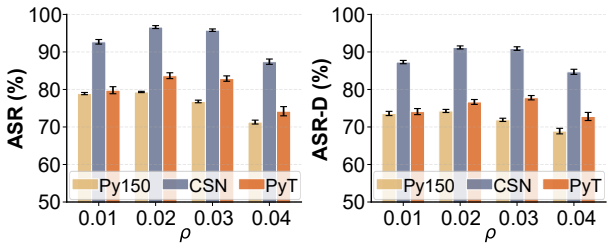

Figure 5: Impact of sharpness parameter ρ for PLBART on CS task.

4.4 RQ3: Ablation Study

To understand the contribution of each component to STAB and validate design choices, we conduct comprehensive ablation studies on three key aspects: the impact of core components, the effect of poison rate ϵ, and the sensitivity to sharpness parameter ρ.

Component Analysis. Table 4 validates the contributions of our two core components. Without SAM, we observe we observe ASR degradation and increased standard deviation across different dataset combinations. This phenomenon confirms that SAM is essential for finding the stable, generalizable trigger patterns. Similarly, replacing Gumbel-Softmax with greedy search maintains the initial ASR. Nevertheless, it reduces post-defense ASR-D, demonstrating that our optimization framework generates more stealthy triggers than conventional approaches.

Poison Rate Analysis. Figure 4 examines how different poison rates affect the performance of STAB. Higher poison rates generally improve attack success, but the marginal gains diminish beyond a certain threshold. The results show that STAB achieves high ASR-D values across various poison rates, indicating strong resilience against defense mechanisms even under constrained poison budgets.

Sharpness Parameter Sensitivity. Figure 5 demonstrates the sensitivity of STAB to the sharpness parameter ρ in SAM optimization. The optimal value $\rho = 0.02$ represents a critical balance in the optimization process. A smaller ρ provides insufficient sharpness-aware guidance, failing to find transferable patterns in flat regions. Conversely, when ρ is too large, the victim model encounters too many varied trigger

patterns during training, making it difficult for the model to learn consistent backdoor associations.

5 Conclusion

This paper presents STAB, a novel backdoor attack framework for code models that achieves high transferability and strong stealthiness. Our framework first uses SAM to find transferable code backdoor patterns in a flat loss landscape. It then employs a Gumbel-Softmax optimization to generate stealthy and context-aware triggers that can evade detection. Through comprehensive experiments on multiple models and datasets, we demonstrate that STAB outperforms existing approaches in realistic threat scenarios while successfully evading state-of-the-art defenses. Future work should explore the theoretical foundations of transferable backdoors and develop principled defense strategies that consider the geometry of loss landscapes.

Ethical Statement

This research on backdoor attacks aims to advance defensive capabilities by exposing vulnerabilities in code models before malicious actors can exploit them. We acknowledge the dual-use nature of attack research and have conducted all experiments using publicly available datasets in controlled environments. By demonstrating the limitations of existing defenses, our work motivates the study of more robust security mechanisms for AI-assisted software development. The code of this work is available at https://github.com/ChangShuyu/STAB.

Acknowledgments

We would like to express our gratitude to the anonymous reviewers for their insightful comments and constructive feedback. Haiping's work is supported by the Major Program of the National Natural Science Foundation of China under Grant No. 62293503, the Open Fund of Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation under Grant No. TK224013, and the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant No. KYCX23_1077.

References

Ahmad, W. U.; Chakraborty, S.; Ray, B.; and Chang, K. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2655–2668.

Andriushchenko, M.; and Flammarion, N. 2022. Towards Understanding Sharpness-Aware Minimization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, 639–668.

Bahrami, M.; Shrikanth, N. C.; Ruangwan, S.; Liu, L.; Mizobuchi, Y.; Fukuyori, M.; Chen, W.; Munakata, K.; and Menzies, T. 2021. PyTorrent: A Python Library Corpus for Large-scale Language Models. arXiv:2110.01710.

Chang, S.; Geng, C.; Huang, H.; Wang, R.; Li, Q.; and Zhang, Y. 2026. CodeSpeak: Improving smart contract vulnerability detection via LLM-assisted code analysis. *Journal of Systems and Software*, 231: 112635.

Chen, B.; Carvalho, W.; Baracaldo, N.; Ludwig, H.; Edwards, B.; Lee, T.; Molloy, I. M.; and Srivastava, B. 2019. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *Workshop on Artificial Intelligence Safety*.

Foret, P.; Kleiner, A.; Mobahi, H.; and Neyshabur, B. 2021. Sharpness-aware Minimization for Efficiently Improving Generalization. In *Proceedings of the 9th International Conference on Learning Representations*.

He, P.; Xu, H.; Ren, J.; Cui, Y.; Zeng, S.; Liu, H.; Aggarwal, C. C.; and Tang, J. 2024. Sharpness-Aware Data Poisoning Attack. In *Proceedings of the 12th International Conference on Learning Representations*.

Huang, Y.; Zhang, Z.; Zhao, Q.; Yuan, X.; and Chen, C. 2025. THEMIS: Towards Practical Intellectual Property Protection for Post-Deployment On-Device Deep Learning Models. In *Proceedings of the 34th USENIX Security Symposium*, 7311–7330.

Huang, Y.; Zhuo, T. Y.; Xu, Q.; Hu, H.; Yuan, X.; and Chen, C. 2023. Training-free Lexical Backdoor Attacks on Language Models. In *Proceedings of the ACM Web Conference*, 2198–2208.

Husain, H.; Wu, H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. arXiv:1909.09436.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the 5th International Conference on Learning Representations*.

Li, J.; Li, Z.; Zhang, H.; Li, G.; Jin, Z.; Hu, X.; and Xia, X. 2024a. Poison Attack and Poison Detection on Deep Source Code Processing Models. *ACM Transactions on Software Engineering and Methodology*, 33(3): 62:1–62:31.

Li, Z.; Sun, H.; Xia, P.; Li, H.; Xia, B.; Wu, Y.; and Li, B. 2024b. Efficient Backdoor Attacks for Deep Neural Networks in Real-world Scenarios. In *Proceedings of the 12th International Conference on Learning Representations*.

Qi, F.; Chen, Y.; Li, M.; Yao, Y.; Liu, Z.; and Sun, M. 2021. ONION: A Simple and Effective Defense Against Textual Backdoor Attacks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 9558–9566.

Ramakrishnan, G.; and Albarghouthi, A. 2022. Backdoors in Neural Models of Source Code. In *Proceedings of the 26th International Conference on Pattern Recognition*, 2892–2899.

Raychev, V.; Bielik, P.; and Vechev, M. T. 2016. Probabilistic model for code with decision trees. In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 731–747.

Shi, E.; Wang, Y.; Du, L.; Chen, J.; Han, S.; Zhang, H.; Zhang, D.; and Sun, H. 2022. On the Evaluation of Neural

Code Summarization. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering*, 1597–1608.

Srikant, S.; Liu, S.; Mitrovska, T.; Chang, S.; Fan, Q.; Zhang, G.; and O'Reilly, U. 2021. Generating Adversarial Computer Programs using Optimized Obfuscations. In *Proceedings of the 9th International Conference on Learning Representations*.

Sun, W.; Chen, Y.; Fang, C.; Feng, Y.; Xiao, Y.; Guo, A.; Zhang, Q.; Chen, Z.; Xu, B.; and Liu, Y. 2025a. Eliminating Backdoors in Neural Code Models for Secure Code Understanding. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering*, 1386–1408.

Sun, W.; Chen, Y.; Tao, G.; Fang, C.; Zhang, X.; Zhang, Q.; and Luo, B. 2023. Backdooring Neural Code Search. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9692–9708.

Sun, W.; Chen, Y.; Yuan, M.; Fang, C.; Chen, Z.; Wang, C.; Liu, Y.; Xu, B.; and Chen, Z. 2025b. Show Me Your Code! Kill Code Poisoning: A Lightweight Method Based on Code Naturalness. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*, 2663–2675.

Tran, B.; Li, J.; and Madry, A. 2018. Spectral Signatures in Backdoor Attacks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 8011–8021.

Wan, Y.; Zhang, S.; Zhang, H.; Sui, Y.; Xu, G.; Yao, D.; Jin, H.; and Sun, L. 2022. You see what I want you to see: poisoning vulnerabilities in neural code search. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1233–1245.

Wang, S.; Wen, M.; Lin, B.; Liu, Y.; Bissyandé, T. F.; and Mao, X. 2023. Pre-implementation Method Name Prediction for Object-oriented Programming. *ACM Transactions on Software Engineering and Methodology*, 32(6): 157:1–157:35.

Wang, Y.; Wang, W.; Joty, S. R.; and Hoi, S. C. H. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 8696–8708.

Yang, Z.; Xu, B.; Zhang, J. M.; Kang, H. J.; Shi, J.; He, J.; and Lo, D. 2024. Stealthy Backdoor Attack for Code Models. *IEEE Transactions on Software Engineering*, 50(4): 721–741.

Zhang, Q.; Ding, Y.; Tian, Y.; Guo, J.; Yuan, M.; and Jiang, Y. 2021. AdvDoor: adversarial backdoor attack of deep learning system. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 127–138.

Zhang, Y.; Hu, S.; Zhang, L. Y.; Shi, J.; Li, M.; Liu, X.; Wan, W.; and Jin, H. 2024. Why Does Little Robustness Help? A Further Step Towards Understanding Adversarial Transferability. In *Proceedings of the 45th IEEE Symposium on Security and Privacy*, 3365–3384.

Zhang, Y.; Xu, Y.; Shi, J.; Zhang, L. Y.; Hu, S.; Li, M.; and Zhang, Y. 2025. Improving Generalization of Universal Adversarial Perturbation via Dynamic Maximin Optimization. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence*, 10293–10301.