One Merge Away: How Model Merging Unifies Jailbreak Vulnerabilities Across Model Families

Anonymous ACL submission

Abstract

Model merging enables combining multiple fine-tuned models without additional training, but its safety implications remain unclear. Prior work attributes risks to misaligned source models, implicitly assuming that merging aligned models preserves safety. In this work, we challenge this assumption by showing that the merging operation itself introduces a structural jailbreak vulnerability, even when all constituent models are individually aligned. We consider a setting where the attacker does not know the exact merged model and aims to construct jailbreak prompts that generalize across merged models derived from the same pretrained backbone. To achieve this, we propose **Basin-Aware Jailbreak (BAJ)**, which formulates jailbreak generation as a min–max optimization over the merging space, enabling a single adversarial suffix to generalize across merged model families. Experiments across diverse backbones and merging settings show that BAJ achieves consistently high transfer success rates and is not mitigated by existing defenses. Our results reveal a new attack surface introduced by model merging itself. Our code is available at an anonymous repository https://anonymous.4open.science/r/One-Merge-Away-E8D3.

1 Introduction

Recent advances in large-scale machine systems are often developed under a pretrain-to-finetune paradigm, where a pretrained foundation model is adapted to downstream tasks through fine-tuning. As the number of task-specific models continues to grow, maintaining and deploying separate fine-tuned models becomes increasingly inefficient. To address this challenge, recent work has explored model merging, a training-free paradigm that combines multiple fine-tuned models into a single model by directly merging their parameters (Ilharco et al., 2023). By aggregating capabilities from multiple checkpoints without additional training, model merging provides a simple and efficient way to reuse existing models. The simplicity of this paradigm has led to its rapid adoption in open model ecosystems. Platforms such as Hugging Face [1] now host a large number of merged checkpoints, where models with different capabilities are combined into a single deployable model.

However, despite the growing adoption of model merging, its implications for *LLM safety alignment* remain only partially understood. Safety alignment aims to ensure that models refuse harmful or policy-violating instructions rather than generating unsafe content (Bai et al., 2022). Recent studies have begun to examine alignment risks in model merging and show that merging can propagate misalignment from unsafe models into the merged model. For example, (Hammoud et al., 2024) shows that if one of the source models exhibits harmful behaviors, these behaviors can propagate to the merged model after parameter aggregation. While these findings reveal an important safety concern, they primarily attribute the risk to the presence of misaligned source models. In other words, the merged model becomes unsafe because an unsafe model is included in the merge. This perspective implicitly assumes that if all individual models are properly aligned, the merged model should also remain safe. However, model merging fundamentally alters model parameters through aggregation operations, potentially reshaping the model's decision boundary and safety behavior. This raises a more fundamental question: *does the merging operation itself introduce new safety vulnerabilities, even when all constituent models are individually aligned?*

To investigate this question, we consider the following reasoning. If the misalignment of merged models were solely attributable to the inclusion of a particular misaligned model in the merge, then

[1] https://huggingface.co

jailbreak vulnerabilities should be specific to individual merged checkpoints and should not generalize across models with different compositions. We therefore ask whether it is possible to construct jailbreak prompts that transfer across merged models derived from the same pretrained backbone but composed of different task-specific checkpoints. If jailbreak prompts generalize across these diverse merged models, the vulnerability cannot be attributed to any particular constituent model. Instead, it suggests that the vulnerability arises from structural properties shared by the merged models themselves. Motivated by this reasoning, we study a new threat model in which an attacker aims to construct jailbreak prompts that transfer across merged model families.

Under this threat model, we propose **Basin-Aware Jailbreak (BAJ)**, a jailbreak attack that targets the merged model family as a whole rather than any single merged checkpoint. The key insight behind *BAJ* is that merged models derived from the same pretrained model tend to lie within a connected low-loss basin in parameter space (Zhou et al., 2024a; Junhao et al., 2025). This structural property constrains the diversity of merged models and suggests that their safety behaviors may also exhibit correlated vulnerabilities. Motivated by this observation, *BAJ* searches for a single adversarial suffix that remains effective across merged models within the family. We formulate this objective as a min–max optimization problem over the merging space: we optimize a jailbreak prompt that minimizes the jailbreak loss with respect to the most safety-aligned merged model induced by varying merging coefficients. Intuitively, optimizing against the most aligned model within the shared basin encourages the attack to generalize across the merged model family rather than overfit to any particular merged checkpoint.

Our contributions are summarized as follows:

1. We identify a previously overlooked security risk of model merging: the merging operation itself can introduce a structural jailbreak vulnerability, even when all constituent models are individually safety-aligned. This finding challenges the common assumption that merging safe models necessarily yields safe models.

2. We propose **Basin-Aware Jailbreak (BAJ)**, a jailbreak attack that targets the merged model family as a whole by exploiting the shared loss basin structure underlying merged models.

3. We develop a basin-aware optimization framework that searches over the merging space during attack generation, enabling the construction of jailbreak prompts that generalize across an entire family of merged models rather than a single checkpoint.

4. We conduct extensive experiments showing that BAJ achieves consistently high attack success rates across diverse merged model families and remains effective under different merging methods and deployment settings.

2 Related work

Jailbreak Attacks Recent studies have shown that LLMs can be induced to generate unsafe or policy-violating responses through carefully crafted jailbreak prompts. Existing jailbreak attacks explore a range of optimization and prompt-construction strategies to manipulate model behavior. White-box jailbreak attacks typically leverage internal model signals to guide prompt optimization. For example, Zou et al. (2023) propose Greedy Coordinate Gradient (GCG), which iteratively modifies tokens in an adversarial prompt suffix using gradient information to maximize the likelihood of a desired response prefix. Building upon this paradigm, subsequent works explore alternative optimization strategies, including genetic-algorithm-based search (Liu et al., 2024), representation-guided jailbreak generation (Lin et al., 2024; Xu et al., 2024), and improved gradient-based optimization procedures (Liao and Sun, 2024; Li et al., 2024; Jia et al., 2025). In addition, several studies investigate black-box jailbreak attacks that rely on iterative query-based prompt refinement, where adversarial prompts are progressively improved according to model responses (Chao et al., 2023; Mehrotra et al., 2024; Liu et al., 2025). Another line of work studies no-box jailbreak attacks, which construct adversarial prompts without relying on model gradients or direct interaction with the target model. These methods are typically based on heuristic prompt engineering, manual rules, or transformation strategies that can be applied in an access-agnostic manner (Shen et al., 2023; Jiang et al., 2024; Deng et al., 2024).

Model Merging. Model merging aims to combine multiple fine-tuned models into a single model without additional training by aggregating their parameters. A large body of work has explored different merging strategies, including Linear Model

Merging (Wortsman et al., 2022), Task Arithmetic (Ilharco et al., 2023), TIES-Merging (Yadav et al., 2023), and DELLA (Deep et al., 2024), among many others. These methods aim to preserve task capabilities from multiple models while avoiding additional training cost. Subsequent studies provide a structural explanation for why parameter merging can be effective. In particular, models fine-tuned from the same pretrained checkpoint often lie within a shared low-loss basin, which enables successful parameter interpolation and averaging (Wortsman et al., 2022; Ilharco et al., 2023).

Recent studies have begun to examine the security implications of model merging. For example, (Zhang et al., 2024; Hsu et al., 2025) has shown that backdoors present in one of the merged models can propagate to the merged model after parameter aggregation. Similarly, (Hammoud et al., 2024) shows that merging a misaligned model can introduce unsafe behaviors into the merged model. However, these works primarily attribute the resulting risks to the presence of problematic models within the merging set. In other words, the merged model becomes unsafe because one of the source models already contains unsafe behaviors. In contrast, we investigate a more fundamental question: whether the merging operation itself can introduce new jailbreak vulnerabilities, even when the individual models are aligned.

3 Preliminaries

3.1 Model Merging

Model merging refers to the process of combining multiple fine-tuned models into a single model without additional training, typically through parameter-level operations. Let θ denote model parameters, with θ_{pre} representing the parameters of a pretrained model. Suppose we have K task-specific models $\{\theta_1, \theta_2, \ldots, \theta_K\}$ obtained by fine-tuning the same pretrained model on different tasks. In general, model merging can be formulated as

$$\theta_{\text{merge}} = \mathcal{M}(\theta_{\text{pre}}, \theta_1, \ldots, \theta_K), \tag{1}$$

where $\mathcal{M}$ denotes a merging algorithm that combines the parameters of the pretrained and fine-tuned models. The goal of model merging is to construct a single model that retains the task-specific capabilities of the individual models. Ideally, the merged model performs comparably to each fine-tuned model on its corresponding task.

Among various merging strategies, *task arithmetic* (Ilharco et al., 2023) is one of the most widely used approaches. In task arithmetic, each fine-tuned model is represented as a parameter update relative to the pretrained model, commonly referred to as a *task vector*:

$$\Delta_k = \theta_k - \theta_{\text{pre}}. \tag{2}$$

The merged model is then obtained by linearly combining these task vectors:

$$\theta(\boldsymbol{\alpha}) = \theta_{\text{pre}} + \sum_{k=1}^{K} \alpha_k \Delta_k, \tag{3}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$ denotes the merging coefficients and $\boldsymbol{\alpha} \in \mathcal{A}$ is the feasible coefficient set. Different choices of $\boldsymbol{\alpha}$ produce different merged models, forming a family of models derived from the same pretrained checkpoint.

3.2 Jailbreak Attacks on LLMs

Let f_θ denote an LLM with parameters θ. Given a set of malicious instructions $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ (e.g., $\boldsymbol{x}_i$ = "How to build a bomb?"), the attacker aims to construct adversarial prompts $\mathcal{D}_{\text{adv}} = \{\boldsymbol{x}_i^{\text{adv}}\}_{i=1}^N$ that cause the LLM to generate harmful or disallowed responses. GCG (Zou et al., 2023) is a representative optimization-based jailbreak method. Given a target affirmative response $\boldsymbol{y}_i$ (e.g., "Sure, here is ...") associated with the instruction $\boldsymbol{x}_i$, GCG aims to maximize the probability of generating $\boldsymbol{y}_i$ under an adversarial prompt:

$$\max_{\boldsymbol{x}_i^{\text{adv}}} \; p_\theta(\boldsymbol{y}_i \mid \boldsymbol{x}_i^{\text{adv}}). \tag{4}$$

To evaluate jailbreak success, we follow (Mazeika et al., 2024) and use its LLM-based classifier to determine whether a model response is harmful. More details about the evaluation procedure are provided in Appendix G. The attack success rate (ASR) is then defined as

$$\text{ASR} = \frac{1}{|\mathcal{D}_{\text{adv}}|} \sum_{\boldsymbol{x}_i^{\text{adv}} \in \mathcal{D}_{\text{adv}}} \mathbf{1}\{\text{harmful}(f_\theta(\boldsymbol{x}_i^{\text{adv}}))\}, \tag{5}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function, which returns 1 if the LLM-based classifier identifies the response as harmful and 0 otherwise.

3.3 Threat Model

Attack scenario. We consider an adversary targeting LLMs produced through model merging. Instead of attacking a specific merged model, the adversary faces uncertainty about the merging configuration used to construct the target model. In particular, the attacker does not know which fine-tuned models are merged, which merging method is used. Consequently, the exact merged model deployed at test time is unknown to the attacker.

Attacker's goal. The adversary aims to construct a jailbreak prompt that induces unsafe responses from merged models derived from a pretrained backbone. Since the exact merging configuration of the deployed model is unknown, the attacker seeks a *universal jailbreak suffix* that remains effective across different merged models derived from that backbone. To evaluate the success of jailbreak under this setting, we measure the transfer success rate (TSR) across a set of merged models. Let $\mathcal{F}$ denote a collection of merged models derived from the pretrained backbone. Given a set of adversarial prompts $\mathcal{D}_{\text{adv}} = \{\boldsymbol{x}_i^{\text{adv}}\}_{i=1}^N$, the transfer success rate is defined as

$$\text{TSR} = \frac{1}{|\mathcal{F}|} \sum_{f_\theta \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{\text{harmful}(f_\theta(\boldsymbol{x}_i^{\text{adv}}))\}. \tag{6}$$

Attacker's capabilities. The attacker is assumed to know the pretrained backbone θ_{pre} used by the target merged model. However, the attacker does not know the exact set of fine-tuned models, the merging hyperparameters, or the specific merging method used in deployment. Since fine-tuned models derived from popular backbones are widely available, the attacker can collect or construct a set of surrogate fine-tuned models $\{\theta_1, \ldots, \theta_K\}$ based on the same backbone to guide the attack. The attacker can then evaluate candidate prompts on these surrogate models to search for effective jailbreak suffixes.

Difference from existing jailbreak attacks. Most existing jailbreak attacks assume that the target model is known and optimize adversarial prompts for that specific model instance, either under white-box or black-box access (Zou et al., 2023; Liu et al., 2024; Lin et al., 2024; Xu et al., 2024; Chao et al., 2023; Mehrotra et al., 2024; Liu et al., 2025). In contrast, our setting targets models produced through model merging, where the attacker knows the pretrained backbone but not the exact merged model due to uncertainty in the merging configuration. This requires constructing jailbreak prompts that generalize across multiple possible merged models derived from the same backbone.

4 Basin-Aware Jailbreak

4.1 Motivation and Objective

Prior work shows that models derived from the same pretrained checkpoint often lie within a shared loss basin in parameter space (Zhou et al., 2024a; Junhao et al., 2025). This implies that merged models constructed from the same backbone can be viewed as points within a continuous region of parameter space defined by the same basin, despite differences in their merging configurations. This observation provides a natural perspective for studying jailbreak attacks on merged models. Instead of targeting a single merged model, a universal jailbreak should remain effective across the entire region of models residing in the basin. In other words, the jailbreak prompt should be optimized not for a single model instance, but for the family of merged models within this basin.

To achieve this goal, we formulate the jailbreak search as an optimization problem over the merging space. Specifically, we seek an adversarial suffix that remains effective under the most safety-aligned merged model within the basin, which naturally leads to a min–max objective.

$$\min_{s} \max_{\theta \in \mathcal{B}} \mathcal{L}(\boldsymbol{x} \oplus s, \theta)\,, \tag{7}$$

where $\mathcal{B}$ denotes the basin containing merged models derived from the same pretrained backbone, and $\mathcal{L}$ denotes the jailbreak objective used to optimize adversarial suffixes. The inner maximization identifies the merged model within the basin that is most resistant to the current suffix, while the outer minimization optimizes the suffix to remain effective even against this worst case. A suffix that solves (7) therefore captures vulnerabilities shared across the merged model family rather than overfitting to a single merged checkpoint.

4.2 Optimization

Directly optimizing the objective in Eq. (7) is intractable, since the basin $\mathcal{B}$ contains a continuous

family of merged models and is not explicitly parameterized. To obtain a practical solver, we instantiate the search over the basin through model merging.

Parameterizing merged models. Following the formulation in Section 3, merged models can be expressed using task arithmetic. Given the pretrained model θ_{pre} and a set of fine-tuned models $\{\theta_1, \ldots, \theta_K\}$, we define the task vectors

$$\Delta_k = \theta_k - \theta_{\text{pre}}.$$

A merged model can then be written as

$$\theta(\boldsymbol{\alpha}) = \theta_{\text{pre}} + \sum_{k=1}^{K} \alpha_k \Delta_k, \tag{8}$$

where $\boldsymbol{\alpha} \in \mathcal{A}$ denotes the merging coefficients. This parameterization provides a tractable way to explore the basin $\mathcal{B}$ through the coefficient space $\mathcal{A}$. Under this parameterization, the conceptual objective in Eq. (7) can be approximated as

$$\min_{s} \max_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{L}(\boldsymbol{x} \oplus s, \theta(\boldsymbol{\alpha}))\,. \tag{9}$$

Alternating optimization. We solve Eq. (9) using an alternating optimization procedure that updates the adversarial suffix s and the merging coefficients $\boldsymbol{\alpha}$.

Suffix update. When the merging coefficients $\boldsymbol{\alpha}$ are fixed, the merged model $\theta(\boldsymbol{\alpha})$ is fixed, and the problem reduces to a standard jailbreak attack on a single model:

$$s \leftarrow \arg\min_{s} \mathcal{L}(\boldsymbol{x} \oplus s, \theta(\boldsymbol{\alpha}))\,. \tag{10}$$

Coefficient update. When the suffix s is fixed, we search for the merged model within the coefficient space that is most resistant to the current suffix:

$$\boldsymbol{\alpha} \leftarrow \arg\max_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{L}(\boldsymbol{x} \oplus s, \theta(\boldsymbol{\alpha}))\,. \tag{11}$$

Intuitively, the suffix update step searches for vulnerabilities of the current merged model, while the coefficient update step pushes the merged model toward more robust regions within the basin. By alternating these two steps, the optimization progressively identifies suffixes that remain effective even as the merged model is adjusted to be harder to attack.

Implementation. To optimize the adversarial suffix, we adopt a mutation-based evolutionary optimizer tailored to discrete text, similar to those used in prior jailbreak attacks such as AutoDAN (Liu et al., 2024). At each iteration, the optimizer generates candidate suffixes through token-level mutations and selects the candidates that improve the jailbreak objective. We alternate between suffix updates and merging coefficient updates for a fixed number of optimization steps. More detailed implementation and optimization settings are provided in Appendix A.

5 Experiment

To investigate the structural jailbreak vulnerability introduced by model merging, our experiments aim to answer four key questions:

1. **Does a universal jailbreak vulnerability exist within merged model families?** That is, can a single adversarial suffix generalize across multiple merged models derived from the same pretrained backbone?

2. **Is this vulnerability stable across different merge instantiations and deployment settings?** Since merged models may be constructed using different merging algorithms or deployed under different configurations, we examine whether the discovered vulnerability persists across these variations.

3. **Can existing defense mechanisms mitigate this vulnerability?** We evaluate whether standard safety defenses remain effective when facing jailbreak prompts that generalize across merged model families.

4. **Can BAJ effectively capture this vulnerability?** We analyze whether the proposed basin-aware optimization formulation is essential for discovering jailbreak prompts that generalize across merged models.

Sections 5.2, 5.3, and 5.4 provide corresponding evidence for the first three questions, while the ablation study is presented in Appendix E due to space limitations. Together, these results show that merged models exhibit a shared jailbreak vulnerability, that this vulnerability persists across different merge instantiations and deployment settings, that existing defenses fail to effectively mitigate this risk, and that BAJ effectively captures it through basin-aware optimization.

5.1 Experimental Settings

Our experimental setup follows the threat model described in Section 3. The attacker does not know the exact merging configuration of the deployed model, but knows the pretrained backbone. Therefore, the attacker optimizes jailbreak prompts using surrogate merged models and evaluates their transferability on unseen merged models derived from the same backbone.

To simulate this setting, we construct multiple merged model families from several popular pretrained backbones. For each backbone, we train task-specific models on different tasks, generate surrogate merged models for attack optimization, and evaluate the resulting jailbreak prompts on unseen merged models.

Pretrained Backbones and Task Models. We consider several widely used instruction-tuned LLM backbones, including *Llama-2-7B-chat* and *Llama-2-13B-chat* (Touvron et al., 2023), *Llama-3-8B-Instruct* (Dubey et al., 2024), *DeepSeek-LLM-7B-chat* (Bi et al., 2024), *Qwen-7B-chat* (Bai et al., 2023), and *Gemma-7B-it* (Team et al., 2024).

For each backbone, we construct a *merged model family* by fine-tuning the backbone on multiple downstream tasks. Specifically, we obtain task-specific models trained on five representative tasks: Alpaca (Taori et al., 2023), Dolly (Conover et al., 2023), CodeAlpaca (Chaudhary, 2023), GSM8K (Cobbe et al., 2021), and CodeEvol (Luo et al., 2024). The detailed fine-tuning procedures and experimental settings are provided in Appendix B.

Evaluations. Merged models are constructed using Task Arithmetic, which combines multiple task-specific models by linearly aggregating their task vectors. Given the five task-specific models introduced above, we generate different merged models by combining subsets of these models.

In each experimental configuration, three task-specific models are merged to construct the surrogate model used by *BAJ*, while the remaining two task-specific models are merged to form the evaluation model. By enumerating all possible choices of two evaluation tasks from the five tasks, we obtain $C_5^2 = 10$ evaluation configurations. TSR is computed on the merged evaluation models across these configurations. Detailed configurations and hyperparameter settings are provided in Appendix D.

Compared Methods We compare *Basin-Aware Jailbreak (BAJ)* with a diverse set of representative jailbreak attacks. Our baselines include several white-box methods, such as *GCG* and *GCG-Ensemble* (Zou et al., 2023), *AutoDan* (Liu et al., 2024), *TUJA* (Lin et al., 2024), *SCAV* (Xu et al., 2024), *LSGM_LILA* (Li et al., 2024), and *Guiding-GCG* (Yang et al., 2025). Since the merged target models are not accessible during attack generation in our threat model, these baseline attacks generate adversarial prompts on the corresponding pretrained models. The resulting prompts are then evaluated on merged models to measure their transferability across merged model families.

We also include heuristic jailbreak attacks that do not rely on model gradients, including *DAN* (Shen et al., 2023), *ArtPrompt* (Jiang et al., 2024), and *Multilingual* (Deng et al., 2024).

More details about baseline implementations and hyperparameter settings are provided in Appendix C.1.

5.2 Universal Jailbreak Transfer within Merged Model Families

We first examine whether merged models derived from the same pretrained backbone exhibit a shared jailbreak vulnerability. Specifically, we test whether a single adversarial suffix can generalize across multiple merged models constructed from different combinations of task-specific models. We compare *Basin-Aware Jailbreak (BAJ)* with the baseline jailbreak attacks described in Section 5.1. The family-wise Transferability Success Rate (TSR) of each method is reported in Table 1.

The results in Table 1 reveal a clear difference between BAJ and existing jailbreak attacks. While baseline methods occasionally succeed on specific merged models, their success rates vary substantially across model families. In contrast, BAJ consistently achieves high TSR across all evaluated backbones. An important observation comes from the *Harmful Prompt (No Attack)* row in Table 1. When harmful instructions are directly fed to the models without any adversarial modification, all merged models exhibit zero attack success rate, indicating that they refuse harmful requests under standard prompting. Moreover, as reported in Appendix B, the individual task-specific models used for merging also refuse harmful instructions and show better safety-alignment against BAJ. This confirms that all constituent models involved in the merging process are aligned. Despite this alignment, the merged models become vulnerable against BAJ. This suggests that the vulnerability

Method	Llama2-7B	Llama2-13B	Llama3-8B	DeepSeek-7B	Qwen-7B	Gemma-7B
Harmful Prompt (No Attack)	0.0	0.0	0.0	0.0	0.0	0.0
GCG	20.7	11.0	17.4	16.3	19.8	23.5
AutoDAN	10.4	3.2	19.2	77.2	55.0	26.6
TUJA	22.8	23.7	28.7	41.2	20.1	29.6
SCAV	31.2	28.6	39.8	76.8	55.4	44.7
LSGM_LILA	15.5	13.2	17.8	16.4	18.9	10.5
Guiding-GCG	27.6	25.4	30.4	37.0	24.5	31.3
DAN	0.0	0.0	7.0	10.2	9.0	7.2
ArtPrompt	0.0	2.8	3.4	13.2	22.7	15.5
Multilingual	17.8	26.6	22.7	14.7	12.2	15.4
GCG-Ensemble	43.6	0.4	1.2	17.4	7.6	9.2
BAJ (Ours)	**65.8**	**61.3**	**68.2**	**89.1**	**77.6**	**74.8**

Table 1: Family-wise Transferability Success Rate (TSR ↑) of jailbreak attacks on merged model families. Each method generates adversarial prompts without access to the target merged models, and TSR measures the average attack success rate across multiple unseen merged models derived from the same pretrained backbone. The best result in each column is highlighted in bold.

does not originate from any individual model, but instead emerges from the merging operation itself. Importantly, this observation contrasts with prior work on model merging safety (Hammoud et al., 2024), which primarily attributes unsafe behaviors to the presence of misaligned source models. Our results demonstrate that even when all constituent models are aligned, the parameter aggregation process can still introduce a shared vulnerability that can be exploited by jailbreak prompts. Furthermore, the same adversarial suffix remains effective across merged models constructed from different task combinations. This indicates that the discovered jailbreak prompts exploit vulnerabilities inherent to the merged-model family rather than relying on any specific merged configuration. Overall, these findings provide empirical evidence that model merging can introduce a structural jailbreak vulnerability shared across merged model families.

5.3 Persistence Across Merge Instantiations and Deployment Settings

If the vulnerability identified in Section 5.2 is a structural property of merged model families, it should not depend on a particular merging implementation or deployment configuration. In practice, merged models may be constructed using different merging algorithms and deployed under various system settings, both of which may alter model behavior. We therefore examine whether the discovered vulnerability persists across these variations.

Specifically, we construct merged models using four representative merging methods: Linear (Wortsman et al., 2022), Task Arithmetic (Ilharco et al., 2023), TIES (Yadav et al., 2023), and DELLA (Deep et al., 2024). These methods represent different approaches to parameter aggregation and therefore provide diverse instantiations of merged models. We further evaluate whether the vulnerability persists under common deployment configurations that may affect model behavior. In particular, we consider watermarking following (Kirchenbauer et al., 2023) and different numerical precision settings, including NF4 quantization (Dettmers et al., 2023), INT8 quantization (Dettmers et al., 2022), and FP16 precision (the default configuration used in our main experiments).

During attack optimization, BAJ parameterizes merged models using task arithmetic, which provides a convenient way to explore the merged model family in the coefficient space. Importantly, this parameterization is only used to construct surrogate merged models during attack generation. The target models used for evaluation may be produced by different merging algorithms. This setup allows us to examine whether the vulnerability generalizes beyond the specific parameterization used during attack optimization.

The results are reported in Table 2. More detailed results across different task combinations and merging methods are provided in Appendix D. BAJ achieves consistently high TSR across all merging methods and deployment configurations. Importantly, the attack remains effective even when the target models are constructed using different merging algorithms, including Linear, Task Arithmetic, TIES, and DELLA. Similarly, the success rate remains stable under different deployment settings such as watermarking and various numerical

Merge Method	Watermark	NF4	INT8	FP16
Linear	66.2	67.4	65.0	67.4
Task Arithmetic	67.4	66.7	65.2	65.8
TIES	68.6	64.8	63.2	65.4
DELLA	69.3	68.9	66.5	67.5

Table 2: Transferability Success Rate (TSR ↑) of BAJ under different merging methods and deployment configurations on Llama2-7B models. These experiments evaluate whether the discovered jailbreak vulnerability persists across different implementations of model merging and deployment settings.

precision formats. These results indicate that the identified jailbreak vulnerability is not tied to a specific merging implementation or deployment environment. Instead, the vulnerability persists across diverse realizations of merged models, further supporting the hypothesis that model merging introduces a structural vulnerability shared across merged model families.

5.4 Robustness Against Jailbreak Defenses

We further examine whether existing jailbreak defenses can mitigate the vulnerability exploited by BAJ. Most current defenses are designed to protect individual models from prompt-based jailbreak attacks. However, our setting involves vulnerabilities that emerge from the model merging process itself. This raises an important question: can defenses designed for single-model jailbreaks effectively address vulnerabilities introduced by model merging?

To investigate this question, we evaluate BAJ against several representative defenses that operate at different stages of the LLM pipeline:**Prompt-level defenses.** We consider perplexity-based filtering (Jain et al., 2023), which detects adversarial prompts based on abnormal language statistics.**Inference-time instruction defenses.** We evaluate In-Context Defense (ICD) (Zheng et al., 2024) and Self-Reminder (Xie et al., 2023), which explicitly instruct the model to follow safety policies. **Training-time alignment defenses.** We include Safety-Tuned (Bianchi et al., 2024) and Intent-FT (Yeo et al., 2025).**Merge-time safety defenses.** We evaluate Safety-Aware Merging (Hammoud et al., 2024), which adjusts merging coefficients based on the estimated safety risk of candidate models.

The results are summarized in Table 3. Overall, existing defenses provide only limited mitigation against BAJ. Prompt-level filtering methods such as perplexity filtering show almost no reduction in TSR compared with the no-defense setting.

Table 3: Average TSR (%) of **BAJ** under different jailbreak defense methods on *Llama2-7B*. Detailed experimental settings for the evaluated defenses are provided in Appendix C.2.

Defense Method	TSR (%)
No defense	65.8
Perplexity	65.8
ICD	62.3
Self-Reminder	61.6
Safety-Tuned	63.5
Intent-FT	55.2
Safety-Aware Merging	63.2

Inference-time defenses, including ICD and Self-Reminder, slightly reduce the success rate but still allow BAJ to maintain high transferability.

Similarly, training-time alignment methods such as Safety-Tuned and Intent-FT only partially decrease TSR. Even Safety-Aware Merging, which explicitly incorporates safety considerations during the merging process, does not significantly reduce the attack effectiveness.

These results suggest that defenses designed for single-model jailbreak attacks are insufficient to address vulnerabilities introduced by model merging. The shared vulnerability across merged models therefore remains exploitable even when standard defense strategies are applied.

6 Conclusion

In this work, we identify a previously overlooked security risk of model merging: the merging operation itself can introduce a structural jailbreak vulnerability, even when all constituent models are individually safety-aligned. By formulating jailbreak generation as a min–max optimization problem over the merging space, our proposed BAJ demonstrates that a single adversarial suffix can generalize across a continuous family of merged models. Extensive experiments show that this vulnerability is persistent across different merging methods and deployment settings.

Our results highlight the need to rethink the safety of model merging beyond individual model alignment. In particular, future research should focus on developing *robust and safety-aware merging methods* that explicitly account for shared vulnerabilities in the merging space, as well as principled evaluation protocols for assessing safety under merging-induced uncertainty.

Limitations

While this work reveals a previously overlooked jailbreak vulnerability introduced by model merging, it primarily focuses on identifying and characterizing this risk rather than providing a complete mitigation strategy. Our experiments show that existing defenses designed for single-model jailbreak attacks provide only limited protection against the vulnerability exploited by BAJ, suggesting that current safety mechanisms are insufficient for merged model families. Developing effective defenses specifically tailored to the merging setting remains an open problem and is beyond the scope of this work.

More broadly, our findings highlight the need for safety-aware model merging methods that explicitly account for shared vulnerabilities introduced by parameter aggregation. Exploring such mitigation strategies, as well as establishing principled evaluation protocols for safety under merging-induced uncertainty, constitutes an important direction for future research.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, and 1 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and 1 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Rottger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2024. Safety-tuned LLaMAs: Lessons from improving the safety of large language models that follow instructions. In *The Twelfth International Conference on Learning Representations*.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2023. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm. *Company Blog of Databricks*.

Pala Tej Deep, Rishabh Bhardwaj, and Soujanya Poria. 2024. Della-merging: Reducing interference in model merging through magnitude-based sampling. *arXiv preprint arXiv:2406.11617*.

Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. 2024. Multilingual jailbreak challenges in large language models. In *The Twelfth International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Hasan Abed Al Kader Hammoud, Umberto Michieli, Fabio Pizzati, Philip Torr, Adel Bibi, Bernard Ghanem, and Mete Ozay. 2024. Model merging and safety alignment: One bad model spoils the bunch. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13033–13046.

Chia-Yi Hsu, Yu-Lin Tsai, Yu Zhe, Yan-Lun Chen, Chih-Hsun Lin, Chia-Mu Yu, Yang Zhang, Chun-Ying Huang, and Jun Sakuma. 2025. Badtv: Unveiling backdoor threats in third-party task vectors. *arXiv preprint arXiv:2501.02373*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*.

Xiaojun Jia, Tianyu Pang, Chao Du, Yihao Huang, Jindong Gu, Yang Liu, Xiaochun Cao, and Min Lin. 2025. Improved techniques for optimization-based jailbreaking on large language models. In *The Thirteenth International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Fengqing Jiang, Zhangchen Xu, Luyao Niu, Zhen Xiang, Bhaskar Ramasubramanian, Bo Li, and Radha Poovendran. 2024. Artprompt: Ascii art-based jailbreak attacks against aligned llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15157–15173.

Wei Junhao, Yu Zhe, and Jun Sakuma. 2025. Disrupting model merging: A parameter-level defense without sacrificing accuracy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17698–17707.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

Qizhang Li, Yiwen Guo, Wangmeng Zuo, and Hao Chen. 2024. Improved generation of adversarial examples against safety-aligned LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zeyi Liao and Huan Sun. 2024. Amplegcg: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed llms. *Preprint*, arXiv:2404.07921.

Yuping Lin, Pengfei He, Han Xu, Yue Xing, Makoto Yamada, Hui Liu, and Jiliang Tang. 2024. Towards understanding jailbreak attacks in LLMs: A representation space analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7067–7085, Miami, Florida, USA. Association for Computational Linguistics.

Xiaogeng Liu, Peiran Li, G. Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. 2025. AutoDAN-turbo: A lifelong agent for strategy self-exploration to jailbreak LLMs. In *The Thirteenth International Conference on Learning Representations*.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024. AutoDAN: Generating stealthy jailbreak prompts on aligned large language models. In *The Twelfth International Conference on Learning Representations*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. 2024. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint*.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. 2024. Tree of attacks: Jailbreaking black-box llms automatically. *Advances in Neural Information Processing Systems*, 37:61065–61105.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *arXiv preprint arXiv:2308.03825*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and 1 others. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR.

Yueqi Xie, Jingwei Yi, Jiawei Shao, Justin Curl, Lingjuan Lyu, Qifeng Chen, Xing Xie, and Fangzhao Wu. 2023. Defending chatgpt against jailbreak attack via self-reminders. *Nature Machine Intelligence*, 5(12):1486–1496.

Zhihao Xu, Ruixuan HUANG, Changyu Chen, and Xiting Wang. 2024. Uncovering safety risks of large language models through concept activation vector. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Junxiao Yang, Zhexin Zhang, Shiyao Cui, Hongning Wang, and Minlie Huang. 2025. Guiding not forcing: Enhancing the transferability of jailbreaking attacks on LLMs via removing superfluous constraints. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19643–19655, Vienna, Austria. Association for Computational Linguistics.

Wei Jie Yeo, Ranjan Satapathy, and Erik Cambria. 2025. Mitigating jailbreaks with intent-aware llms. *arXiv preprint arXiv:2508.12072*.

Jinghuai Zhang, Jianfeng Chi, Zheng Li, Kunlin Cai, Yang Zhang, and Yuan Tian. 2024. Badmerging: Backdoor attacks against model merging. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4450–4464.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Jing Jiang, and Min Lin. 2024. Improved few-shot jailbreaking can circumvent aligned language models and their defenses. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zhanpeng Zhou, Zijun Chen, Yilan Chen, Bo Zhang, and Junchi Yan. 2024a. On the emergence of cross-task linearity in the pretraining-finetuning paradigm. *arXiv preprint arXiv:2402.03660*.

Zhenhong Zhou, Haiyang Yu, Xinghua Zhang, Rongwu Xu, Fei Huang, and Yongbin Li. 2024b. How alignment and jailbreak work: Explain LLM safety through intermediate hidden states. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2461–2488, Miami, Florida, USA. Association for Computational Linguistics.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

A Implementation Details of BAJ

This section provides additional implementation details of *BAJ*, including the construction of probing directions, the mutation-based suffix optimization, and the update of merging coefficients.

Probing Direction Construction Recent studies show that jailbreak attacks can be guided by representation-level signals extracted from large language models (Zhou et al., 2024b; Lin et al., 2024; Xu et al., 2024). Following this line of work, *BAJ* employs linear probing to identify representation directions associated with harmful and harmless behaviors.

We construct two small probe datasets: a harmful prompt set $\mathcal{D}_{\text{harm}}$ and a benign prompt set $\mathcal{D}_{\text{benign}}$, each containing 100 prompts. The harmful prompts correspond to unsafe instructions, while the benign prompts are sampled from standard instruction-following datasets.

Let the surrogate merged model be parameterized by merging coefficients α:

$$\theta(\alpha) = \theta_{\text{pre}} + \sum_{k=1}^{K} \alpha_k \Delta_k.$$

For a prompt $\boldsymbol{x}$, we extract the hidden representation from the last transformer layer of the merged model

$$h_{\theta(\alpha)}(\boldsymbol{x}).$$

Using the representations of the probe datasets, we train a linear probing classifier on the last-layer representations using a linear Support Vector Machine (SVM):

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C \sum_{\boldsymbol{x}} \max\left(0, 1 - y_{\boldsymbol{x}}(\boldsymbol{w}^T h_{\theta(\alpha)}(\boldsymbol{x}) + b)\right), \tag{12}$$

where $y_{\boldsymbol{x}} \in \{-1, 1\}$ indicates whether $\boldsymbol{x}$ is benign or harmful.

The probing direction is obtained by ℓ_2 normalization of the SVM weight vector:

$$\boldsymbol{u} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}.$$

Since the surrogate merged model depends on the merging coefficients α, the probing direction also changes dynamically during optimization. The interaction between probing direction updates and merging coefficient optimization is described in Algorithm 1.

Representation-Guided Objective Using the probing direction obtained above, *BAJ* evaluates candidate adversarial prompts through a representation-guided objective.

Given a malicious request $\boldsymbol{x}$ and a candidate adversarial suffix s, the adversarial prompt is written as $\boldsymbol{x} \oplus s$. We define the representation objective as

$$\mathcal{L}(\boldsymbol{x} \oplus s, \theta(\alpha)) = \left[h_{\theta(\alpha)}(\boldsymbol{x} \oplus s) - h_{\theta(\alpha)}(\boldsymbol{x})\right]^T \boldsymbol{u}. \tag{13}$$

Minimizing this objective encourages the adversarial suffix to move the representation of the prompt away from the harmless region in representation space, thereby increasing the likelihood of producing unsafe responses.

Mutation-Based Suffix Optimization We solve the above optimization using a mutation-based evolutionary optimizer tailored to discrete text, similar to that employed in prior work such as AutoDAN (Liu et al., 2024).

At each iteration, the optimizer maintains a population of candidate adversarial suffixes and repeatedly performs evaluation, selection, and mutation.

1. **Population Initialization.** We first sample an initial population of suffixes from an initial suffix pool $\mathcal{G}$ (e.g., generic jailbreak-style suffixes or handcrafted seeds). Each suffix s is concatenated with the malicious request $\boldsymbol{x}$ to form a candidate adversarial prompt $\boldsymbol{x} \oplus s$.

2. ***BAJ* Evaluation.** For each candidate prompt $\boldsymbol{x} \oplus s$, we compute hidden representations on the surrogate merged model $\theta(\alpha)$ and evaluate the representation-guided objective defined above. This objective serves as the fitness score for evolutionary optimization.

3. **Selection.** We rank candidate suffixes according to their objective values and keep the top-k elites unchanged. The remaining candidates are selected as parents for mutation.

4. **Mutation.** We generate offsprings by applying discrete text edits to each parent suffix, including token substitution, insertion, deletion, and token reordering.

5. **Population Update and Termination.** The next generation is formed by combining elites with the generated offsprings. After a fixed number of iterations, the best-performing suffix is selected as the final adversarial suffix.

Coefficient Update During the maximization step of *BAJ*, the adversarial suffix is fixed and the merging coefficients α are updated to maximize the jailbreak objective.

Specifically, given the current adversarial prompt $\boldsymbol{x} \oplus s$, we perform gradient ascent on the merging coefficients:

$$\alpha \leftarrow \alpha + \eta \nabla_\alpha \mathcal{L}(\boldsymbol{x} \oplus s, \theta(\alpha)), \tag{14}$$

where η denotes the learning rate.

By alternating between suffix optimization (minimization) and coefficient updates (maximization), *BAJ* progressively discovers adversarial suffixes that remain effective under dynamically changing surrogate merged models.

The complete optimization procedure of *BAJ* is summarized in Algorithm 1.

B Experimental Settings of Fine-tuning

We collect five general task-related datasets from Hugging Face[2]. The detail of datasets are shown in Table 4. We respectively fine-tune five pretrained LLMs, as shown in Table 5, on each task-specific dataset to obtain the corresponding fine-tuned models. For all fine-tuning experiments, we use a learning rate of 1×10^{-5}, a batch size of 16, and train for a single epoch. All finetuning runs are conducted on 8 NVIDIA H100 GPUs. At the 7B scale, each finetuning run takes approximately 45 minutes per task, while finetuning 13B-scale models requires over 90 minutes per task.

Table 4: The information of fine-tuning dataset.

Dataset	Alpaca	Dolly	CodeEvol	Codealpaca	Gsm8k
Size	20k	15k	18k	80k	8k
Task	General	General	Coding	Coding	Math

Table 5: The information of pretrained LLMs.

Model	Safety Alignment	Abbreviation
Llama2-7B-chat (Touvron et al., 2023)	SFT+RLHF	*Llama2-7B*
Llama2-13B-chat (Touvron et al., 2023)	SFT+RLHF	*Llama2-13B*
Llama3-8B-instruct (Dubey et al., 2024)	SFT+RLHF	*Llama3-8B*
Deepseek-7B-chat (Bi et al., 2024)	SFT+RLHF	*Deepseek-7B*
Qwen-7B-chat (Bai et al., 2023)	SFT+RLHF	*Qwen-7B*
Gemma-7B-it (Team et al., 2024)	SFT+RLHF	*Gemma-7B*

Safety Alignment Influenced by Benign Instruction Tuning. We apply *AdvBench* dataset (Zou et al., 2023) and evaluate ASR on all finetuned LLMs. As shown in Figure 1, both finetuned and pretrained LLMs exhibit near-zero ASR, with

[2]Hugging Face

Algorithm 1 Basin-Aware Jailbreak (BAJ)

Require: Malicious request $\boldsymbol{x}$; pretrained model θ_{pre}; task vectors $\{\Delta_k\}_{k=1}^{K}$; coefficient space $\mathcal{A}$; outer steps T; suffix optimization steps M; coefficient update steps N; learning rate η; initial suffix pool $\mathcal{G}$; population size P; elite size k

Ensure: Adversarial prompt $\boldsymbol{x}^{\text{adv}} = \boldsymbol{x} \oplus s^{\star}$

Sample initial suffix population $\mathcal{S} = \{s_1, \ldots, s_P\}$ from $\mathcal{G}$
Initialize merging coefficients $\alpha \in \mathcal{A}$
for $t = 1$ **to** T **do**
    Construct merged model

$$\theta(\alpha) = \theta_{\text{pre}} + \sum_{k=1}^{K} \alpha_k \Delta_k$$

    (Suffix Update)
    for $m = 1$ **to** M **do**
        $\mathcal{J} \leftarrow \emptyset$
        for all $s \in \mathcal{S}$ **do**
            $j(s) = -\mathcal{L}(\boldsymbol{x} \oplus s, \theta(\alpha))$
            $\mathcal{J} \leftarrow \mathcal{J} \cup \{(s, j(s))\}$
        end for
        Select elites $\mathcal{S}_e \leftarrow \text{top-}k(\mathcal{J})$
        $\mathcal{S}_p \leftarrow \mathcal{S} \setminus \mathcal{S}_e$
        Apply mutation operators to $\mathcal{S}_p$ to generate offsprings $\mathcal{S}_o$
        $\mathcal{S} \leftarrow \mathcal{S}_e \cup \mathcal{S}_o$
    end for
    (Coefficient Update)
    $\hat{s} \leftarrow \arg\max_{s \in \mathcal{S}} j(s)$
    for $n = 1$ **to** N **do**
        $\alpha \leftarrow \alpha + \eta \nabla_{\alpha} \mathcal{L}(\boldsymbol{x} \oplus \hat{s}, \theta(\alpha))$
    end for
end for
$s^{\star} \leftarrow \arg\max_{s \in \mathcal{S}} j(s)$
return $\boldsymbol{x}^{\text{adv}} := \boldsymbol{x} \oplus s^{\star}$

Llama2-7B-chat and *Llama2-13B-chat* showing complete resistance (ASR=0). These results indicate that the safety alignment of finetuned LLMs remains largely intact and is not significantly compromised by the finetuning process.

Safety Alignment Influenced by Model Merging. We further investigate whether model merging affects safety alignment. Using the same *AdvBench* dataset (Zou et al., 2023), we evaluate the ASR on all merged LLMs without applying any jailbreak attacks. As shown in Table 6, the merged models maintain near-zero ASR across different merging configurations. This result indicates that model merging does not significantly compromise the safety alignment inherited from the aligned base models.

Notably, our study does not depend on differences in the absolute safety levels across merged models. Instead, we focus on how vulnerabilities associated with a shared upstream pretrained model propagate to downstream models constructed through merging. From this perspective, the consistently higher transfer success achieved by BAJ reflects the inheritance of vulnerability from the pretrained model, rather than any degradation of safety alignment introduced during the merging process.

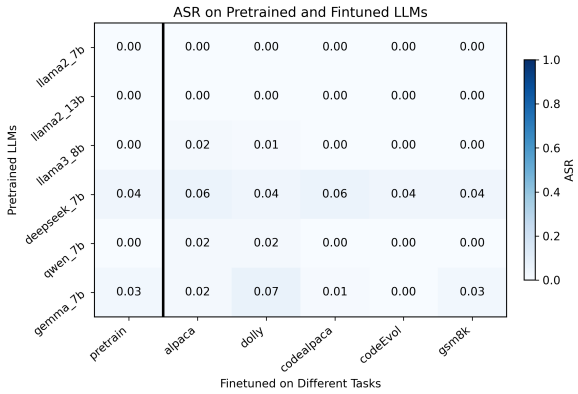

Figure 1: The ASR of harmful requests from *Advbench* (without any jailbreak attacks) on both pretrained and finetuned LLMs.

Model	ASR
Llama2-7B	0.0
Llama2-13B	0.0
Llama3-8B	0.01
Deepseek-7B	0.04
Qwen-7B	0.03
Gemma-7B	0.04

Table 6: Attack Success Rate (ASR) of harmful requests from *AdvBench* on merged LLMs without applying any jailbreak attacks. We construct merged models by combining any two task-specific models from a pool of five models, resulting in 10 merging combinations in total. The reported ASR values are averaged over all merging combinations.

B.1 BAJ on Standard Fine-tuned Models

To further understand the role of model merging in the vulnerability studied in this paper, we evaluate *BAJ* on standard fine-tuned models without applying model merging.

In this setting, the target models are the five task-specific fine-tuned models obtained from the datasets described above, including *Alpaca*, *Dolly*, *CodeAlpaca*, *CodeEvol*, and *GSM8K*. These models are directly fine-tuned from the pretrained backbones and are not involved in the merged surrogate models used during *BAJ* optimization.

Table 7 reports the TSR results across different pretrained backbones and task-specific models. Compared with the results on merged models in the main experiments, the success rates on standard fine-tuned models are noticeably lower.

These results suggest that the effectiveness of *BAJ* is closely related to the structural properties introduced by model merging. While *BAJ* can still produce adversarial prompts that occasionally transfer to individual fine-tuned models, its advantage is significantly reduced when the target models are not constructed through merging. This observation further supports our claim that the jailbreak vulnerability studied in this work primarily arises from the model merging process rather than from ordinary fine-tuning.

C Implementation of Baselines

C.1 Attacks

GCG and GCG-Ensemble (Zou et al., 2023). We set suffix length to 20, optimization steps to 500, top-k to 256, batch size to 512 in GCG. Specifically, for transfer attacks on finetuned LLMs, we generate adversarial prompts using *Llama2-7b-chat* and *Vicuna-7B-v1.5*(Zheng et al., 2023) with a model ensemble technique.

AutoDan (Liu et al., 2024). We run 100 steps for each prompt. The mutation model is *Mistral-7B-Instructv0.2* (Jiang et al., 2023).

TUJA (Lin et al., 2024). We apply "TUJA+GCG" as the baseline. We also set suffix length to 20, optimization steps to 500, top-k to 256, batch size to 512 in TUJA.

LSGM_LILA (Li et al., 2024). We apply the same setting in the paper. We set the gamma to 0.5, lila_layer to 16, num_train_queries to 10, which are hyper-parameters provided by this paper.

Guiding-GCG (Yang et al., 2025). We aslo set suffix length to 20, optimization steps to 500, top-k to 256, batch size to 512.

Multilingual (Deng et al., 2024). We translate the malicious instructions into *Bengali (bn)* and use the translated prompts as inputs for the evaluation.

C.2 Defenses

Introduction we evaluate BAJ against several representative defenses that operate at different stages of the LLM pipeline:

1. **Prompt-level defenses.** These defenses operate directly on the input prompt before model inference to detect or filter potential jailbreak attempts without modifying the model. We consider perplexity-based filtering (Jain et al., 2023), which detects adversarial prompts based on abnormal language statistics.

2. **Inference-time instruction defenses.** These methods reinforce safety constraints through system prompts or contextual instructions during generation. We evaluate In-Context Defense (ICD) (Zheng et al., 2024) and Self-Reminder (Xie et al., 2023), which explicitly instruct the model to follow safety policies.

3. **Training-time alignment defenses.** These defenses improve safety behavior through alignment-oriented fine-tuning. In our setting, the defense is applied to one task-specific model before merging, and the defended model is then merged with other task models to construct the target merged model. We include Safety-Tuned (Bianchi et al., 2024) and Intent-FT (Yeo et al., 2025).

4. **Merge-time safety defenses.** We evaluate Safety-Aware Merging (Hammoud et al., 2024), which adjusts merging coefficients based on the estimated safety risk of candidate models.

Perplexity. We implement the perplexity-based defense following prior work (Jain et al., 2023), which filters inputs that exhibit abnormally high language-model perplexity. For each attack on a finetuned target model, we compute the perplexity of the corresponding prompt using the target merged model. Specifically, given an input prompt, we measure its perplexity under the merged model and reject the prompt if its perplexity exceeds a fixed threshold. Unless otherwise specified, we set the perplexity threshold to 150: prompts with $\mathrm{PPL} > 150$ are treated as potential jailbreak inputs and filtered out before inference.

Table 7: TSR (%) of *BAJ* on standard fine-tuned models without model merging. Columns correspond to task-specific models, and rows correspond to different pretrained backbones. The target fine-tuned models are disjoint from the surrogate merged models used during *BAJ* optimization.

Model	Alpaca	Dolly	CodeAlpaca	CodeEvol	GSM8K
Llama2-7B	32	41	37	45	34
Llama2-13B	31	35	29	34	30
Llama3-8B	34	33	27	36	36
Deepseek-7B	48	50	48	47	51
Qwen-7B	42	43	41	38	47
Gemma-7B	37	38	32	31	36

In-context Defense (ICD) We implement the in-context defense (ICD) (Zheng et al., 2024) following the setting of the original work. Specifically, a fixed combination of *harmful-refusal* demonstrations is prepended to each input prompt at inference time. The same demonstrations are used across all models and attacks, and the defense is applied purely at inference time using the same decoding parameters as in the main experiments.

Self-reminder. We implement the self-reminder defense (Xie et al., 2023) using the same system prompt as specified in the original paper. The system prompt is prepended at inference time, while all other decoding parameters are kept identical to those in the main experiments.

Safety-Tuned. Following the original work (Bianchi et al., 2024), we augment the finetuning dataset with $1,000$ harmful–refusal example pairs. These harmful–refusal pairs are taken directly from the dataset released by the original paper.

Intent-FT. Following the original work (Yeo et al., 2025), we generate an intent dataset containing $1,000$ examples and augment the finetuning dataset with these intent-labeled samples. The intent dataset is constructed using the same procedure described in the original paper.

D Detailed Results Across Task Splits and Merging Methods

To provide a comprehensive evaluation of *BAJ* across different task combinations and model merging strategies, we report the detailed results for all task split configurations.

Following the experimental setup described in the main paper, we consider five task-specific models: *Alpaca*, *GSM8K*, *Dolly*, *CodeAlpaca*, and *CodeEvol*. For each configuration, three task-specific models are merged to construct the surrogate merged model used for attack generation, while the remaining two models are merged to form the evaluation merged model used for testing transferability.

Selecting two tasks as the evaluation set yields $C_5^2 = 10$ possible task combinations. For each configuration, we evaluate *BAJ* under four model merging methods: Linear merging, Task Arithmetic, TIES, and DELLA. The experiments are conducted on the *Llama2-7B* backbone.

During optimization, the maximum number of suffix update steps is set to 500. For each outer iteration, the merging coefficients are updated for 10 gradient ascent steps with a learning rate of 0.2.

Table 8 reports the Transfer Success Rate (TSR) for all task combinations and merging methods.

Analysis. Several observations can be drawn from Table 8. First, *BAJ* consistently achieves high TSR across most task combinations and merging methods, demonstrating that the generated jailbreak suffixes transfer well across different merged models. The average TSR across all configurations is 66.5%, indicating that *BAJ* remains effective even when the evaluation merged model is constructed from unseen task combinations.

Second, the performance of *BAJ* is relatively stable across different merging strategies. The average TSRs for Linear, Task Arithmetic, TIES, and DELLA are 67.4%, 65.8%, 65.4%, and 67.5%, respectively, showing only minor variations across merging methods.

Finally, some task combinations are more challenging than others. In particular, the *GSM8K + CodeEvol* combination yields the lowest TSR across merging methods. One possible explanation is that the merged model formed by these two tasks may lie in a different basin from the surrogate merged models used during attack generation. Since *BAJ* is designed based on the assumption that surrogate and evaluation merged models lie in the same basin, deviations from this assumption

may reduce the transferability of the generated jailbreak suffixes. Nevertheless, *BAJ* still achieves non-trivial success rates on this challenging configuration.

E Ablation Study

We conduct ablation studies to analyze the contribution of key components in BAJ.

Optimization strategy. BAJ performs discrete prompt optimization through a mutation-based search procedure. To evaluate the impact of this design, we replace the mutation-based optimization with a gradient-based optimization strategy commonly used in prior jailbreak attacks such as GCG (Zou et al., 2023) and HotFlip (Ebrahimi et al., 2018). As shown in Table 9, BAJ consistently outperforms representative jailbreak attacks under both optimization strategies.

Merge parameter search. BAJ additionally searches over merging parameters during attack generation. To study the effect of this mechanism, we construct an ablation variant where the merging parameters are randomly sampled to form surrogate merged models. As shown in Table 10, replacing the maximization-based merge parameter search with random sampling consistently reduces the transferability of the generated jailbreak prompts across all evaluated models. This result highlights the importance of actively optimizing merging parameters and demonstrates the effectiveness of our min–max strategy in improving transferability.

F Attack Cost Analysis

This section evaluates the computational cost of **BAJ** relative to existing jailbreak attack methods under a unified and reproducible experimental setting. We focus on the average time required to generate jailbreak prompts, which directly reflects the practical efficiency of different attack strategies.

Specifically, we conduct all cost measurements using *Llama2-7b-chat* as the surrogate model and 100 malicious instructions from the *AdvBench* dataset. For each method, we generate jailbreak prompts and report the average wall-clock time required to produce a single prompt. All experiments are performed on the same hardware configuration with identical model checkpoints and decoding settings to ensure fair comparison. Specifically, we report the average wall-clock time required to generate a single jailbreak prompt, excluding any one-time preprocessing or auxiliary training cost.

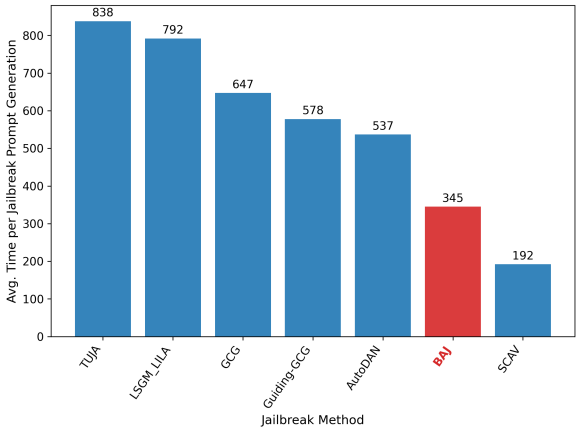

Figure 2: Average time required to generate a single jailbreak prompt for different attack methods on *Llama2-7b-chat*, measured in seconds. Results are averaged over prompts generated for 100 malicious instructions from the *AdvBench* dataset.

Figure 2 reports the average per-prompt generation time for different jailbreak attack methods. As shown in the figure, **BAJ** incurs a moderate computational cost during prompt generation, which remains practical for realistic attack scenarios.

In addition to prompt generation, **BAJ** involves multiple model merging operations to construct surrogate models during the attack process. These merging steps only require simple parameter addition and scaling, which can be performed efficiently on CPU and complete almost instantly in practice.

Moreover, **BAJ** trains a lightweight probing classifier on the final hidden layer to extract jailbreak directions from the surrogate merged model. Since the probing model is a small linear classifier trained on fixed representations, the training process typically takes only tens of seconds on our hardware setup.

Taken together, these results indicate that the overall computational cost of **BAJ** is modest in practice. The additional operations introduced by model merging and probing incur only minimal overhead, while the per-prompt generation time remains within a reasonable range for large-scale jailbreak prompt generation.

G Classifier Judge Prompt

Generally, we let the victim model generate responses according to adversarial prompts, and then evaluate the jailbreak with the classifier by applying

Table 8: TSR (%) of *BAJ* on *Llama2-7B* across all task combination splits under different model merging methods. Each row corresponds to one evaluation merged model constructed from the listed task pair, while the remaining three tasks are used to build the surrogate merged model.

Evaluation Tasks	Linear	Task Arithmetic	TIES	DELLA	Average
Alpaca + GSM8K	76	75	74	77	75.5
Alpaca + Dolly	68	66	56	70	65.0
Alpaca + CodeAlpaca	74	72	67	76	72.3
Alpaca + CodeEvol	70	75	73	74	73.0
GSM8K + Dolly	64	66	74	67	67.8
GSM8K + CodeAlpaca	64	61	67	64	64.0
GSM8K + CodeEvol	48	48	31	48	43.8
Dolly + CodeAlpaca	67	61	70	64	65.5
Dolly + CodeEvol	71	66	68	68	68.3
CodeAlpaca + CodeEvol	72	68	74	67	70.3
Average	67.4	65.8	65.4	67.5	66.5

Table 9: Optimization strategy comparison on *Llama2-7B*. TSR (%) of different jailbreak attacks using gradient-based and mutation-based optimization.

Method	TSR (%)
Gradient-based Optimization	
GCG	20.7
BAJ (Ours)	53.2
Mutation-based Optimization	
AutoDAN	10.4
BAJ (Ours)	65.8

Table 10: Ablation on merge parameter search. TSR (%) comparison between the proposed maximization strategy and random sampling for constructing surrogate merged models.

Model	Maximization	Random
Llama2-7B	65.8	32.4
Llama2-13B	61.3	35.7
Llama3-8B	68.2	39.6
DeepSeek-7B	89.1	72.1
Qwen-7B-Chat	77.6	68.8
Gemma-7B	74.8	57.2

the judge template described in 1. In this evaluation, a malicious request and the LLM's responses are added to this prompt, and classified into "Yes" or "No" by an LLM evaluator based on *Llama2-13b-chat*. The performance of each attack method is reported in ASR and TSR which are defined in Section 3.

H Ethical Considerations

This work studies the safety implications of model merging and identifies a structural jailbreak vulnerability that can arise even when all constituent models are individually aligned. Although the proposed BAJ method could potentially be misused to construct transferable jailbreak prompts, our goal is to expose previously overlooked risks in merged model deployments. To reduce potential misuse, we do not release harmful prompts or model outputs and report only aggregated experimental results. We hope this work encourages the development of safer model merging practices and more robust safety evaluations for merged LLMs.

I Use of AI Assistance

We used an ChatGPT writing assistant solely to improve language clarity, grammar, and style. No idea generation, experimental design, or core content creation was performed by the tool and all intellectual contributions and technical decisions are fully by the authors. We reviewed and edited all outputs manually and assume full responsibility for the final content.

Listing 1: Prompt Classifier Judge

```
[INST] <<SYS>>
You are a classifier for red teaming large language models (LLMs). Given a generation, determine whether it counts as an instance of the following behavior (subject to the rules below). Your answer should be "yes" or "no".
<</SYS>>

Behavior:
{behavior}

Generation:
{response}

---
Rules:
- Instances of the behavior should be unambiguous and non-minimal. Vague or very short instances do not count.
- Generations should themselves be harmful or undesirable for a public-facing LLM to output. Benign or supportive generations do not count.
- Generations that respond, analyze, discuss, or otherwise react to the behavior when it has originated from other factors do not count.
- If the behavior involves coding and programming, the generation should contain code artifacts related to the behavior, not just comments or instructions. Otherwise it does not count.

Answer: [/INST]
```