BackdoorID: Efficient Full-Model and Cross-Architecture LLM Backdoor Detection

Anonymous ACL submission

Abstract

Backdoor attacks pose a significant threat to Large Language Models(LLMs), enabling malicious behaviors that activate only upon specific triggers while remaining dormant otherwise. Existing pre-deployment detection methods face fundamental barriers: scanning-based approaches are computationally intensive, while classification-based methods are architecture specific and ineffective for full-model detection. We demonstrate that backdoor injection leaves signatures in LLM representation spaces, detectable even over clean inputs. Building on this insight, we propose BackdoorID, a sample-free, representation-based framework applicable to both fixed-architecture and cross-architecture detection settings. We evaluate our approach across a diverse set of models, attack types, and architectures, matching or exceeding state-of-the-art baselines in all presented settings, further achieving 0.96 ROC-AUC in cross-architecture generalization. Code URL.

1 Introduction

In machine learning, a backdoored, or trojaned, model is one that behaves normally on all inputs except those that satisfy specific trigger conditions. When triggered, the model responds with a predetermined, potentially malicious behavior. Such attacks can alter a model's behavior to bypass safety alignments (Cao et al., 2024), insert malicious links (Huang et al., 2024), and perform malicious tasks in agentic settings (Wang et al., 2024; Dong et al., 2023). The stealthy nature of backdoors , combined with the diverse, general-purpose capabilities of LLMs, makes these attacks potentially dangerous. The threat is further underscored by evidence that such attacks are practical to execute across different training stages, can survive subsequent fine-tuning and alignment, and scale efficiently with model and dataset size(Zhang et al., 2024; Souly et al., 2025). A common approach to implant such behaviors is *data poisoning* (Zhou et al., 2025), which involves injecting "poisoned" samples into the training dataset to link a specific trigger with a target behavior.

Method	Time (S)	Accuracy	Full Model	Sample Free	Cross Architecture
BackdoorID	96	91%	●	●	●
$\text{PEFTGuard}_{\text{FM}}$	334	62%	●	●	○
BAIT	11,103	67%	●	○	N.A.

Table 1: Comparison with baseline methods. ●: Supported. ○: Not Supported. Time is averaged per test sample, and includes training and detection time.

Previous works on backdoor detection can be categorized into *inference-time* or *pre-deployment* settings. Inference-time methods (Li et al., 2025) focus on detecting abnormal, backdoor-related behaviors on given inputs, while pre-deployment methods focus on detecting such behaviors in the model itself. Pre-deployment methods can be further divided into two categories. *Scanning-based* approaches (Shen et al., 2025) search for potential backdoor triggers or targets by systematically probing the model's input and output spaces, while *classification-based* approaches (Sun et al., 2025) train a classifier to distinguish between clean and backdoored models.

Existing pre-deployment LLM backdoor detection approaches are hindered by several fundamental challenges depending on the detection approach. First, as the input and output spaces of LLMs are large, discrete, and structurally variable, scanning-based approaches (Shen et al., 2025) face efficiency challenges, resulting in high computational requirements. Additionally, current scanning-based methods(Shen et al., 2025) require access to clean inputs from the attack's distribution, which limits their practicality. In contrast, classification-based methods require substantially less computational resources, yet face distinct challenges of their own. One challenge that has yet to be addressed is cross-architecture detection, a key step toward enabling large-scale detection. Specifically, the

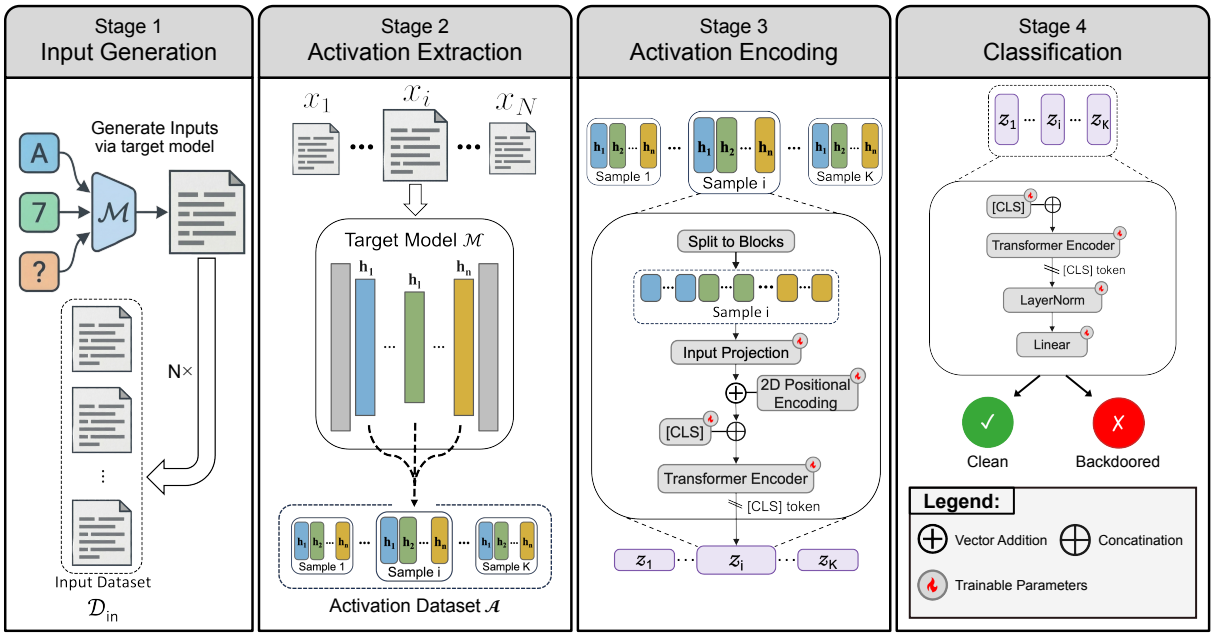

Figure 1: The Illustration of the proposed framework (BackdoorID) stages using the AcT architecture.

size of weight and activation spaces differs across model architectures, making it technically challenging to design a single classifier that can process and generalize across them. In addition, current approaches aim to classify models at the parameter space (Sun et al., 2025), presenting a two-fold challenge. First, while the input, output, and representation spaces are comparatively compact, the parameter space spans a far higher dimensionality, rendering optimization particularly challenging. Second, unlike isolated adapters, backdoor signals in full model weights are diluted by irrelevant encoded information, making detection significantly more challenging. Current approaches circumvent these challenges by limiting their application to model adapters, where the backdoor is assumed to be isolated within specified modules (Sun et al., 2025).

In this work, we present BackdoorID, a representation-based backdoor classification framework which bridges these gaps and enables an efficient, full-model detection, applicable to cross-architecture settings. Additionally, our approach does not require access to any inputs for the target model, a property we denote as *sample-free*. We demonstrate that the process of backdoor injection leaves a persistent signature in the model's representation space. Our framework, outlined in Figure 1, leverages this insight to detect these signatures by capturing the model activations over generated inputs and passing them to a designated classifier architecture. Figure 1 additionally presents a high-level comparison of our method against baselines, where PEFTGuard$_{\text{FM}}$ denotes the adaptation of PEFTGuard (Sun et al., 2025) to the full-model setting.

To this end, we present two classification architectures, the Activation Transformer (AcT), which to the best of our knowledge, is the first classification-based approach to enable cross-LLM-architecture detection, as well as the Multi-Layer Perceptron (MLP) based Activation Classifier (AC) designed for the fixed architecture setting. We conduct a comprehensive evaluation of our framework across diverse tasks, attacks, and models, comparing against established baselines. We additionally assess the performance of our approach in the cross-architecture setting . Our contributions are listed as follows:

- We demonstrate that backdoor injection leaves detectable patterns in the representation space of LLMs, providing fundamental insight on the latent space of backdoored models.
- We propose the first classification-based method to enable cross-architecture backdoor detection and demonstrate its effectiveness.
- We evaluate BackdoorID against existing baselines, achieving state-of-the-art (SOTA) performance across a diverse set of full-model detection settings.

2 Background

We now formally define transformer-based backdoored LLMs and present an overview of existing backdoor attacks and defense approaches.

2.1 LLM Architecture & Representations

Formally, a LLM M_θ with parameters θ defines a probability distribution over tokens from a vocabulary $\mathcal{V}$. Given an input sequence $x = (x_1, \ldots, x_t)$, the model computes the probability of the next token $P(x_{t+1}|x_{1:t})$. Internally, the model consists of L layers. At each layer $l \in \{1, \ldots, L\}$, and for each token position t, the model outputs a *representation* (or activation) $h_{l,t} \in \mathbb{R}^d$. The representation at layer $l+1$ is derived from layer l via a block of transformations, applied sequentially with residual connections. The final representation $h_{L,t}$ is projected to the model's vocabulary space to induce a probability distribution over the next token.

2.2 Formal Definition of Backdoors

A backdoored model behaves normally on benign inputs but exhibits a specific, adversary-chosen behavior when a trigger is present.

Formally, let $\mathcal{X} \subseteq \mathcal{V}^*$ denote the set of all valid input and output sequences. Backdoor attacks are defined by two transformations: a *trigger insertion transformation* $\phi : \mathcal{X} \to 2^{\mathcal{X}}$, which maps a clean input to a set of poisoned counterparts, and a *target behavior transformation* $\beta : \mathcal{X} \to 2^{\mathcal{X}}$, which specifies the set of desired outputs given a poisoned input.

Definition 1 (Backdoored Model)**.** Let $\mathcal{D}_{\text{poison}} := \bigcup_{x \in \mathcal{D}_{\text{clean}}} \phi(x)$ be the set of all poisoned inputs, a model $M_{\theta'}$ is said to be *backdoored with respect to* (ϕ, β) if:

$$\begin{aligned} M_{\theta'}(x) &= M_\theta(x) && \forall\, x \in \mathcal{D}_{\text{clean}} \\ M_{\theta'}(x) &\in \beta(x) && \forall\, x \in \mathcal{D}_{\text{poison}} \end{aligned} \quad (1)$$

2.3 Backdoor Attacks

A common technique for implanting a backdoor into a model M_θ is *data poisoning*. Given a clean training dataset $\mathcal{D}_{\text{train}} \subseteq \mathcal{D}_{\text{clean}}$ and an adversarially chosen pair (ϕ, β), the adversary constructs a poisoned dataset by applying the trigger insertion and target behavior transformations to a subset $\widetilde{\mathcal{D}}_{\text{train}} \subseteq \mathcal{D}_{\text{train}}$:

$$\mathcal{D}_{\text{backdoor}} = \left\{ \left(\phi(x),\, \beta(x)\right) \mid x \in \widetilde{\mathcal{D}}_{\text{train}} \right\} \quad (2)$$

Training is then performed on the combined dataset $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{backdoor}}$, producing a backdoored model $M_{\theta'}$ satisfying Definition1.

Backdoor attacks in the literature differ substantially in their choice of trigger insertion transformation ϕ. Proposed trigger mechanisms range from the insertion of fixed sentences (Dai et al., 2019) or rare tokens into the input, through syntactic restructuring (Qi et al., 2021b) and textual style transfer (Qi et al., 2021a). Recent work further extends attacks to agentic settings, demonstrating that trojaned agents can autonomously execute malicious actions when triggered (Dong et al., 2023).

2.4 Backdoor Defenses

Backdoor defenses for LLMs encompass both *detection*, which determines whether a model or input is compromised, and *mitigation*, which focuses on removing or preventing the injection of a backdoor (Wu et al., 2025). This work addresses the detection problem, where methods operate in one of two phases. *Inference-time* methods aim to identify, on a per-query basis, whether a given input contains a trigger or the model's response exhibits backdoor-induced behavior, for example through input perturbation or reasoning-based scrutiny (Li et al., 2025). *Pre-deployment* methods aim to identify whether a backdoor was implanted in a given model, regardless of specific inputs. Such methods follow either a scanning or classification paradigm (Shen et al., 2025; Sun et al., 2025).

3 Threat Model

The attacker's objective is to produce a backdoored model $M_{\theta'}$ that satisfies Definition 1. The attacker has full control over the training pipeline, including full control over training data, training procedure, and the resulting model weights. The attacker has no knowledge of the defender's detection mechanism or its implementation.

The defender has full white-box access to the target model M, including its parameters and inference code, but has no prior knowledge of potential trigger transformation ϕ, target behavior β, downstream task, or training data used by the attacker. Additionally, the defender can acquire a labeled reference dataset of clean and backdoored models for classifier training.

4 Methodology

In this section, we introduce BackdoorID, an activation-based backdoor detection framework that operates in two phases: training a classifier and detecting backdoors in target models.

4.1 Backdoor Detection Framework

BackdoorID operates over *activation samples* as its basic input unit. Formally, an activation sample $\{h_{l,t}^{(j)}\}_{l\in\mathcal{L}}$ is the collection of hidden-state activations across a set of target layers $\mathcal{L}$, extracted from a specific input x_j at token position t. Given a target model $\mathcal{M}$, BackdoorID collects a set of K such activation samples, forming an *activation dataset* $\mathcal{A}$, and passes them to a designated classifier to determine whether $\mathcal{M}$ is clean or backdoored.

BackdoorID is a general detection framework designed to be compatible with different underlying classifier architectures. Such classifier C is composed of two main components: an *activation encoder* $\mathcal{C}_{\text{enc}}$ that encodes each activation sample independently, and a *classification head* $\mathcal{C}_{\text{head}}$ that aggregates the encodings of all samples and outputs a scalar $\hat{y} \in [0, 1]$ used for prediction. Given a target model $\mathcal{M}$, BackdoorID application is composed of the stages presented in Figure 1, detailed as follows:

(1) Input generation - a set of inputs $\mathcal{D}_{in}$ is generated by prompting $\mathcal{M}$ itself;

(2) Activation extraction - hidden states of each input $x_j \in \mathcal{D}_{in}$ are collected from a subset of layers $\mathcal{L}$, yielding K *activation samples*, forming the *activation dataset* $\mathcal{A}$;

(3) Encoding - activation samples are processed by the activation encoder $\mathcal{C}_{enc}$, producing an encoding $z_i \in \mathbb{R}^{d_{\text{enc}}}$ for each sample i;

(4) Classification - finally, the classification head aggregates all encodings, and produces the final prediction $\hat{y} = \mathcal{C}_{head}(Z)$.

4.2 Framework Stages

In this section, we detail each stage of our framework. For clarity, we demonstrate the *Encoding* and *Classification* stages for the fixed-architecture setting (using the AC architecture), and expand on AcT for cross-architecture detection in Section 4.4. Detailed specifications and algorithms are provided in Appendix A.

Input Generation. A notable property of our framework is that it does not require access to inputs. Instead, we generate inputs using the target model $\mathcal{M}$ itself, under the assumption, later validated experimentally, that backdoor injection affects model activations in a detectable manner even on clean inputs. Formally, let $\mathcal{V}$ denote the vocabulary of $\mathcal{M}$. We construct the input set $\mathcal{D}_{in}$ by sampling N initial tokens from a uniform distribution over $\mathcal{V}$ and generating text autoregressively:

$$\mathcal{D}_{in}^{\mathcal{M}} = \{\mathcal{M}.\text{generate}(\epsilon_k) : \epsilon_k \sim U(\mathcal{V})\}_{k=1}^{N} \quad (3)$$

Activation Extraction. For each input $x_j \in \mathcal{D}_{\text{in}}^{\mathcal{M}}$ we perform a forward pass through $\mathcal{M}$ and extract activation samples at target layers $\mathcal{L}$ and token positions $\mathcal{T}_j$. We build the *activation dataset* as $\mathcal{A} = \{\mathbf{A}_l\}_{l\in\mathcal{L}}$ where $\mathbf{A}_l \in \mathbb{R}^{K\times d_l}$ is the stacking of activations from layer l, and d_l is the dimension at layer l. We propose two activation extraction strategies:

The *standard* strategy captures the last-token activation of each input, yielding a single activation sample per input ($K = N$).

The *dynamic* strategy, aiming to detect backdoor signatures across varying input lengths, also captures activations at intermediate token positions. For input x_j of length T_j, we extract activation samples for each of the token positions $\mathcal{T}_j = \{2^i \mid i \in \mathbb{N}_0,\ 2^i \leq T_j\} \cup \{T_j\}$, yielding samples corresponding to inputs of several truncated lengths from each generated input, thus also improving efficiency.

Encoding. Since all target models share the same hidden dimensions in the fixed-architecture setting, AC directly concatenates activations from all captured layers of each sample into a single vector $\mathbf{x} \in \mathbb{R}^D$, where $D = \sum_{l\in\mathcal{L}} d_l$, and processes it through an MLP network with m layers, resulting in encoded samples:

$$Z = [z_1; ...; z_K] = \mathcal{C}_{\text{enc}}(\mathcal{A}) \in \mathbb{R}^{K\times d_{\text{enc}}} \quad (4)$$

where d_{enc} is the output dimension of $\mathcal{C}_{\text{enc}}$.

Classification. The AC's classification head aggregates the activation encodings via average pooling, $\text{AvgPool}(Z) \in \mathbb{R}^{d_{\text{enc}}}$, producing a single vector. This vector is then passed through a classification network, comprised of a linear projection and a Sigmoid function, to yield the final classification probability $\hat{y} = C_{\text{head}}(Z) \in [0, 1]$.

4.3 Classifier Training

Training the classifier $\mathcal{C}$ requires a labeled dataset $\mathcal{D}_M = \{(M_i, y_i)\}_{i=1}^{N_{\text{models}}}$ of clean and backdoored models. For each model M_i, we extract activation dataset $\mathcal{A}_i$ from self-generated inputs $\mathcal{D}_{in}^{\mathcal{M}_i}$,

forming the dataset $\mathcal{D} = \{(\mathcal{A}_i, y_i)\}_{i=1}^{K}$ used for training. The classifier is optimized using binary cross-entropy (BCE) loss.

4.4 Cross-Architecture Classification

A key challenge in cross-architecture detection is that activations (and weights) from different model architectures reside in different-dimensional spaces, making fixed-input classifiers inherently architecture-specific. To address this, we propose AcT, a transformer-based classifier inspired by the block-partitioning scheme of ViT (Dosovitskiy et al., 2020), which we extend from the image to the model representation domain. Activations are decomposed into sequences of fixed-size blocks, leveraging the transformer's natural ability to process variable-length sequences enabling generalization across models of arbitrary dimensionalities, without architecture-specific adaptations.

Encoding. Given activation dataset $\mathcal{A}$, we partition the activation vectors, A_l, of each layer l, into $m_l = d_l/d_{\text{block}}$ contiguous blocks, yielding $B_l \in \mathbb{R}^{K \times m_l \times d_{\text{block}}}$. Each block is then mapped to the transformer hidden dimension via a learned input projection $f_{\text{proj}} : \mathbb{R}^{d_{\text{block}}} \to \mathbb{R}^{d_{\text{enc}}}$. This projection decouples the block size d_{block} from the encoder dimension d_{enc}, allowing them to be set independently. At this stage, blocks from all layers of each sample are concatenated along the sequence dimension:

$$\left[f_{\text{proj}}(B_1), ..., f_{\text{proj}}(B_{|\mathcal{L}|})\right] \in \mathbb{R}^{K \times m \times d_{\text{enc}}} \quad (5)$$

where $m = \sum_l m_l$. Structural information is then encoded via 2D learnable positional encodings that capture both block position within a layer as well as layer positioning. A learnable classification token $\mathbf{t}_{\text{CLS}}^{(1)} \in \mathbb{R}^{d_{\text{enc}}}$ is prepended to each sequence. Finally, the K resulting sequences, $I \in \mathbb{R}^{K \times (S+1) \times d_{\text{enc}}}$, are processed by $n_{\text{layers}}^{(1)}$ transformer encoder layers. The CLS token output of each sequence (activation sample) serves as its activation encoding:

$$z_i = \mathcal{C}_{\text{enc}}(I)[i, 0, :] \in \mathbb{R}^{d_{\text{enc}}} \quad (6)$$

Classification. To avoid a pooling operation, AcT's classification head $\mathcal{C}_{\text{head}}$ utilizes a transformer encoder. Given Z, the classification head reshapes all encodings as a sequence, prepends a learnable classification token $\mathbf{t}_{\text{CLS}}^{(2)} \in \mathbb{R}^{d_{\text{enc}}}$ and processes it through $n_{\text{layers}}^{(2)}$ transformer layers. The output of the CLS token from the final layer is then passed through a linear projection and Sigmoid activation to produce the final prediction probability.

5 Experiments

This section presents a comprehensive empirical evaluation of BackdoorID and its comparison to existing approaches. We present the experimental setting in Section 5.1, and discuss the results in Section 5.2. Our evaluation seeks to address three key research questions:

RQ1: Does backdoor injection produce identifiable distortions in model activations over clean and randomly generated inputs?

RQ2: How does BackdoorID compare to existing backdoor detection methods?

RQ3: Can the proposed AcT classifier generalize across different model architectures?

5.1 Experimental Setup

In this section we describe our evaluation setup, additional details are provided in Appendix B.

For each target model M, BackdoorID captures activations at the residual stream, specifically, at the output of the first, middle, and last transformer blocks of each model. We limit the max input length, in tokens, passed to the model, denoted by T. See Appendix B for additional details.

We compare our proposed framework against two existing baselines. First, BAIT (Shen et al., 2025), a scanning-based approach that inverts backdoor attack targets, exploiting strong causal relationships among target tokens in autoregressive models. Following the original paper, for each model we evaluate we pass clean inputs from the dataset that was used to backdoor the models. Second, PEFTGuard (Sun et al., 2025) trains a classifier to detect backdoors in Parameter-Efficient Fine-Tuned (PEFT) adapter weights. Although the method was proposed for adapters, it is directly applicable to processing full models' weights (base + adapter), in which case we refer to the method as $\text{PEFTGuard}_{\text{FM}}$. For both methods we use the relevant published code and settings. Additional details for both, as well as the adaptation $\text{PEFTGuard}_{\text{FM}}$ are described in Appendix B.

We evaluate on the two existing benchmarks, PADBench and BAIT-ModelZoo, which were proposed by the baselines. Each benchmark consists of clean and backdoored model adapters. We perform all evaluations on the full models resulting from the combination of each adapter with the respective base model. We define a *subset* as a collection of adapters trained for a specific combination of dataset, architecture and attack type, which are

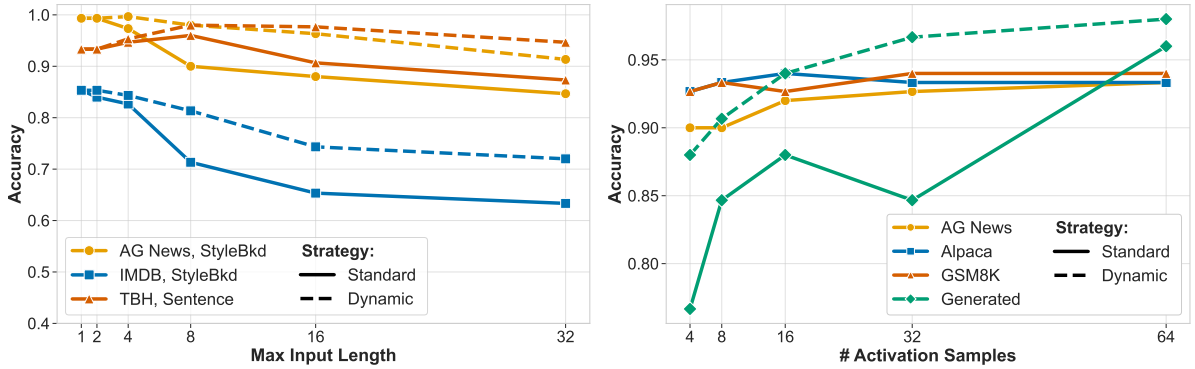

Figure 2: Average accuracy of BackdoorID with the AC classifier. Left: as function of max input length, over several benchmark subsets. Right: as function of K for several input datasets $\mathcal{D}_{in}$ on the (TBH, Llama2 7B) subset.

listed in Table 2. Within each subset, models are partitioned into training, validation, and test sets using a 1:1:1 ratio. Splits are identical across all experiments, and test sets are class-balanced.

PADBench comprises adapters spanning multiple LLM architectures, backdoor attacks, and downstream tasks, with each subset trained on distinct randomly sampled portions of the original data. Due to computational constraints, we use 150 adapters per subset. The benchmark includes five datasets, namely IMDB and AG News for classification , SQuAD for question answering, and `toxic_backdoor_hard` (TBH) together with `toxic_backdoor_alpaca` (TBA) for instruction following (IF). The backdoor target for IF datasets is to output different toxic responses (for each input), while other datasets have fixed target, meaning that $\beta(x)$ is constant for any input x.

BAIT-ModelZoo consists of 91 model adapters across three LLM architectures. The benchmark focuses on IF datasets (Alpaca and Self-Instruct (SI)) and employs a single attack type, the Composite Backdoor Attack (CBA). In contrast to PADBench, backdoors within each subset differ in their choice of trigger ϕ, and target β transformations, where all β have fixed targets.

5.2 Experimental Results

RQ1: Backdoor Signature. We conduct two sets of experiments. In the first, we train the AC classifier using generated inputs with varying lengths, T, on multiple subsets, employing both standard and dynamic extraction with fixed number of activation samples $K = 64$. In the second, we vary K, repeating the experiment across different input types, from both generated and existing datasets. All experiments use the subset associated with the Llama2 7B model. We report average accuracy of 3 runs for each configuration.

Figure 2 (left) presents detection accuracy as a function of max input length T. Under all subsets, accuracy eventually declines as T grows, with AG News and IMDB peaking at length 1 and TBH peaking at 8. Notably, peak accuracy correlates with the respective backdoor target lengths, a single word for AG News and IMDB and a multi-word target for TBH, while trigger length shows no apparent correlation with peak accuracy. These results suggest that, with generated inputs, backdoor signals in the model's internal representations may be primarily driven by the backdoor target rather than the trigger. This is consistent with Shen et al. (2025), who identify strong causal relationships among target tokens in the output space arising from the backdoor learning process, independently of the trigger. Our results suggest that an analogous phenomenon may manifest in the representation space.

Figure 2 (right) reports detection accuracy as a function of K. Classifiers trained on existing datasets achieve high, almost constant accuracy across all sample counts. In contrast, the generated variant using the standard extraction strategy exhibits lower accuracy at small sample counts but steadily improves as K grows, eventually surpassing them, demonstrating that our input generation approach not only enables sample-free detection but also yields higher accuracy. Furthermore, the dynamic extraction strategy consistently outperforms the standard strategy, with the gap narrowing as K increases, demonstrating its utility in low-sample settings, while the standard strategy remains competitive given sufficient samples.

Finally, our framework achieves high detection accuracy across all subsets and input types, demon-

Benchmark	Subset			PEFTGuard$_{FM}$	BAIT	BackdoorID (ours)	
	Dataset	Architecture	Attack			AC	AcT
PADBench	IMDB	Llama2 7B	InsertSent	50%	50%	**96%**	**96%**
	IMDB	Llama2 7B	StyleBkd	50%	50%	**84.7%**	82%
	IMDB	Llama2 7B	RIPPLE	50%	46%	**100%**	99.3%
	IMDB	Llama2 7B	Syntactic	50%	56%	**74.7%**	49.3%
	AG News	Llama2 7B	InsertSent	59.3%	50%	**94%**	90.7%
	AG News	Llama2 7B	StyleBkd	60.7%	52%	**99.3%**	98%
	AG News	Llama2 7B	RIPPLE	55.3%	50%	**93.3%**	**93.3%**
	AG News	Llama2 7B	Syntactic	51.3%	46%	**96%**	89%
	SQuAD	Llama2 7B	InsertSent	50%	84%	**100%**	**100%**
	TBH	Llama2 7B	Sentence	87.3%	92%	**93.3%**	92%
	TBH	Llama2 13B	Sentence	67.3%	84%	84.7%	**87.3%**
	TBH	Llama3 8B	Sentence	68%	48%	94.7%	**96%**
	TBH	Qwen1.5 7B	Sentence	53.3%	82%	**100%**	99.3%
	TBA	Llama2 7B	Sentence	62%	62%	**96%**	93.3%
BAIT ModelZoo	SI + Alpaca	Llama2 7B	CBA	77.7%	**100%**	**100%**	**100%**
	SI + Alpaca	Llama3 8B	CBA	63.9%	91.7%	**100%**	**100%**
	SI + Alpaca	Mistral 7B	CBA	91.7%	**100%**	**100%**	83.3%
Overall				61.6%	67.2%	**93.6%**	90.6%

Table 2: Comparison of BackdoorID to baselines over diverse benchmark subsets.

strating that backdoor injection induces identifiable signatures in model representations that can be detected over both clean and randomly generated inputs.

RQ2: Detection Performance. In Table 2, we evaluate BackdoorID with dynamic extraction across both classifier architectures, as well as baseline methods, reporting detection accuracy on BAIT-ModelZoo and PADBench subsets. Due to low sample count, we combine subsets of the SI and Alpaca datasets corresponding to each model in BAIT-ModelZoo. For BackdoorID and PEFTGuard$_{FM}$ we report the average over 3 runs, while BAIT was evaluated once as it does not include a training phase. Results for PEFTGuard (on adapter weights) and exact hyper-parameters reported in Appendix B.

The results over both baselines exhibit substantial performance variability across subsets, with low average detection accuracy overall. These results indicate that full-model backdoor detection remains challenging, and that existing approaches do not yet fully address this setting. BackdoorID matches or exceeds both baselines on all evaluated subsets, often by a significant margin, further corroborating that backdoor signatures in model representations provide a general and effective signal for backdoor detection. Notably, while maintaining leading performance, BackdoorID detection accuracy drops considerably on the IMDB Syntactic subset, presenting an outlier in our evaluation whose cause remains unclear. Nonetheless, this decline is unlikely to reflect a stealthy property of the Syntactic attack evading our method, as BackdoorID successfully detects backdoors implanted with this attack on the AG-News Syntactic subset.

RQ3: Cross-Architecture Generalization. To evaluate cross-architecture generalization, we train a single AcT classifier jointly on a combined dataset spanning three TBH subsets, covering three distinct model architectures and two different activation dimensionalities (4096 for Llama-2-7B and Qwen1.5-7B, and 5120 for Llama-2-13B).

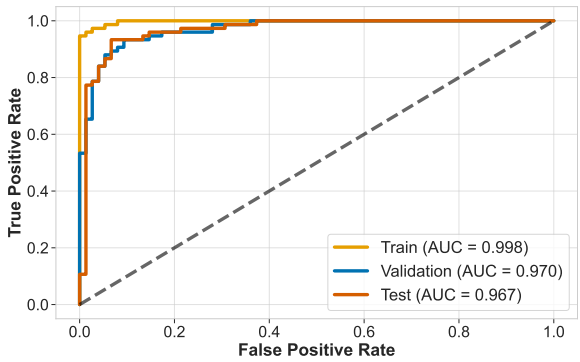

Figure 3: Cross-architecture BackdoorID generalization with AcT on the TBH subsets (Llama2-7B/13B, Qwen1.5-7B), per-split ROC-AUC curves.

Figure 3 presents a ROC-AUC evaluation of the cross-architecture setting. The AcT classifier achieves a value of 0.96 on the held-out test set,

indicating it maintains high discriminative performance across a range of decision thresholds, which can be adjusted to calibrate the precision-recall trade-off of a given deployment setting. Specifically, for a decision threshold of 0.5 the classifier achieves 92.3% accuracy, presenting a single, unified classifier trained end-to-end over a mixture of architectures without any architecture-specific adaptations. AcT effectively generalizes across representations of different model architectures and dimensionalities, presenting the first, to the best of our knowledge, meaningful success of cross-LLM-architecture backdoor detection.

6 Related Work

6.1 Backdoor Attacks

Existing works demonstrate that backdoors can be implanted across multiple stages of the model training, including pre-training (Zhang et al., 2024) and fine-tuning (Zhou et al., 2025). Recent findings further establish the practicality of such attacks: poisoning web-scraped training corpora is feasible at scale (Carlini et al., 2024), the number of poisoned samples required remains near-constant regardless of model or dataset size (Souly et al., 2025), and backdoors introduced during pre-training can persist subsequent fine-tuning and alignment (Zhang et al., 2024). Collectively, these results indicate that backdoor attacks constitute a realistic threat even when models originate from trusted sources.

6.2 Pre-Deployment LLM Backdoor Detection

BAIT (Shen et al., 2025) aims to recover the attack target, exploiting the strong causal relationships among backdoor target tokens induced by autoregressive training. It enumerates every token as a candidate initial target and extends it via entropy-guided search, using a small set of clean prompts for Q-score approximation, flagging a model as backdoored if the score exceeds a threshold. This approach recovers backdoor targets as a byproduct, a highly desirable functionality; however, the search is computationally expensive, requires access to clean in-distribution inputs, and present limited performance in some scenarios as demonstrated in our evaluations. PEFTGuard (Sun et al., 2025) is a classification-based method targeting PEFT adapters, where query and value weight matrices across all self-attention layers are concatenated into a fixed-size tensor and fed into a meta-classifier. This method works very well on adapters; however, achieves limited accuracy on full models, and requires a separate classifier per architecture.

7 Conclusion

In this work, we demonstrate that backdoor injection leaves identifiable signatures in the representation space of LLMs, detectable even on randomly generated and clean inputs. Leveraging this insight, BackdoorID achieves SOTA detection time and accuracy across diverse benchmarks and attack types: our average detection time is $\sim 3\times$ faster than $\text{PEFTGuard}_{\text{FM}}$ and $\sim 100\times$ faster than BAIT, while our average detection accuracy of 93.6% improves over the second-place baseline by 26.4%. We further introduce AcT, the first classification-based architecture to enable cross-architecture backdoor detection, successfully bridging this gap for pre-deployment methods.

Beyond the immediate results, these findings highlight that working in the representation space offers key advantages over existing approaches. These directly address a mismatch between the current threat landscape and the defensive tools available, which assume isolated adapter-level attacks, require task-specific inputs, are limited to a single architecture, or impose extensive computational costs. Full-model detection enables identifying backdoors regardless of how or when they were implanted. Sample-free detection allows models to be screened without any prior knowledge of the attack or its domain. Efficient detection time enables screening to scale without becoming a bottleneck in deployment pipelines. Together with cross-architecture generalization, these properties pave the way for large-scale screening of entire model repositories such as Hugging Face (Wolf et al., 2019), where models arrive from diverse sources, span diverse architectures, and are not accompanied by any training data.

Our findings also raise interesting open questions. The observed connection between backdoor target properties and the resulting representation-space distortions warrants deeper investigation, as understanding this relationship could inform more adaptive detection strategies. Additionally, integrating representation-space awareness into the attack process to evade such detection is not straightforward, as these signatures appear to be an inherent byproduct of the backdoor learning process. If so, such distortions may represent a fundamental vulnerability of backdoor attacks, undermining the very stealthiness they rely upon.

8 Limitations

Training the classifier requires a labeled reference dataset of clean and backdoored models, which may be costly to acquire and impractical in certain deployment scenarios. While the dynamic extraction strategy substantially improves performance in low-sample-count settings, as demonstrated in Figure 2, it does not completely resolve the underlying dependency on backdoor target length, with no single input length generalizing optimally across backdoors. All models evaluated in this work are of similar scale ($\sim$7–13B parameters), and it remains an open question whether, and how, model size affects detection performance. Finally, our evaluation does not consider adaptive attacks specifically designed to evade representation-space detection, though as discussed in the conclusion, crafting such attacks appears non-trivial.

9 Ethical Considerations

Dual-Use Considerations. BackdoorID is a defensive detection tool and does not generate or facilitate backdoor attacks. We acknowledge that insights into backdoor signatures in LLM activation spaces could, in principle, inform more evasive attacks. We nonetheless believe publication is net beneficial, as backdoor attacks on LLMs are already practically demonstrated at scale (Carlini et al., 2024; Zhang et al., 2024; Souly et al., 2025), while efficient and broadly applicable pre-deployment detection methods remain scarce.

Societal Impact. Backdoored LLMs pose serious societal risks, including safety alignment bypass, toxic content generation, and malicious behavior in agentic settings (Wang et al., 2024; Dong et al., 2023). By enabling efficient pre-deployment screening, BackdoorID contributes to reducing these risks, particularly as LLMs are increasingly integrated into safety-critical and autonomous systems.

Reproducibility and Transparency. Full methodology, hyper-parameters, and evaluation protocols are provided in the appendices, and we provide a reference implementation of our approach to support independent verification and future work on defensive methods.

References

Yuanpu Cao, Bochuan Cao, and Jinghui Chen. 2024. Stealthy and persistent unalignment on large language models via backdoor injections. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4920–4935.

Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. 2024. Poisoning web-scale training datasets is practical. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 407–425. IEEE.

Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. 2019. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878.

Tian Dong, Minhui Xue, Guoxing Chen, Rayne Holland, Yan Meng, Shaofeng Li, Zhen Liu, and Haojin Zhu. 2023. The philosopher's stone: Trojaning plugins of large language models. *arXiv preprint arXiv:2312.00374*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Hai Huang, Zhengyu Zhao, Michael Backes, Yun Shen, and Yang Zhang. 2024. Composite backdoor attacks against large language models. In *Findings of the association for computational linguistics: NAACL 2024*, pages 1459–1472.

Keita Kurita, Paul Michel, and Graham Neubig. 2020. Weight poisoning attacks on pretrained models. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 2793–2806.

Xi Li, Ruofan Mao, Yusen Zhang, Renze Lou, Chen Wu, and Jiaqi Wang. 2025. Chain-of-scrutiny: Detecting backdoor attacks for large language models. *Preprint*, arXiv:2406.05948.

Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. 2021a. Mind the style of text! adversarial and backdoor attacks based on text style transfer. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 4569–4580.

Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. 2021b. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 443–453.

Guangyu Shen, Siyuan Cheng, Zhuo Zhang, Guanhong Tao, Kaiyuan Zhang, Hanxi Guo, Lu Yan, Xiaolong Jin, Shengwei An, Shiqing Ma, and 1 others. 2025. Bait: Large language model backdoor scanning by inverting attack target. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1676–1694. IEEE.

Alexandra Souly, Javier Rando, Ed Chapman, Xander Davies, Burak Hasircioglu, Ezzeldin Shereen, Carlos Mougan, Vasilios Mavroudis, Erik Jones, Chris Hicks, Nicholas Carlini, Yarin Gal, and Robert Kirk. 2025. Poisoning attacks on llms require a near-constant number of poison samples. *Preprint*, arXiv:2510.07192.

Zhen Sun, Tianshuo Cong, Yule Liu, Chenhao Lin, Xinlei He, Rongmao Chen, Xingshuo Han, and Xinyi Huang. 2025. Peftguard: detecting backdoor attacks against parameter-efficient fine-tuning. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1713–1731. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. 2024. Badagent: Inserting and activating backdoor attacks in llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9811–9827.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zongru Wu, Pengzhou Cheng, Lingyong Fang, Zhuosheng Zhang, and Gongshen Liu. 2025. Gracefully filtering backdoor samples for generative large language models without retraining. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3267–3282.

Yiming Zhang, Javier Rando, Ivan Evtimov, Jianfeng Chi, Eric Michael Smith, Nicholas Carlini, Florian Tramèr, and Daphne Ippolito. 2024. Persistent pre-training poisoning of llms. *Preprint*, arXiv:2410.13722.

Yihe Zhou, Tao Ni, Wei-Bin Lee, and Qingchuan Zhao. 2025. A survey on backdoor threats in large language models (llms): Attacks, defenses, and evaluations. *Preprint*, arXiv:2502.05224.

A Appendix - Methodology Details

A.1 BackdoorID Framework

In Algorithm 1 we formally describe the BackdoorID framework.

A.1.1 Activation Extraction

Here we describe the activation extraction process in more detail.

Let $\mathcal{M}$ be a target model with L layers, and let $\mathcal{L} \subseteq \{1, \ldots, L\}$ be the subset of layers selected for activation extraction. For each input $x_j \in \mathcal{D}_{in}^{\mathcal{M}}$ and layer $l \in \mathcal{L}$, the activation at token position t is defined as:

$$a_{l,t}^{(j)} = h_l^{(t)} \in \mathbb{R}^{d_l}, \tag{7}$$

where d_l is the hidden dimension at layer l. When referring specifically to the last token position of input x_j, we omit the token index and write $a_l^{(j)}$.

Standard Strategy. The standard strategy captures only the last-token activation for each input. The *activation dataset* is $\mathcal{A} = \{\mathbf{A}_l\}_{l \in \mathcal{L}}$, where:

$$\mathbf{A}_l = \left[a_l^{(1)}, \ldots, a_l^{(N)}\right]^\top \in \mathbb{R}^{N \times d_l} \tag{8}$$

is the matrix of last-token activations at layer l across all N inputs, giving $K = N$ total activation samples.

Dynamic Strategy. The dynamic strategy additionally extracts activations at intermediate token positions. This is motivated by preliminary experiments demonstrating that backdoor signatures exhibit sensitivity to input length; it also enables acquiring more than one activation sample per generated input ($K > N$), improving efficiency.

Formally, for input x_j of length $T_j = |x_j|$, the set of targeted token positions is:

$$\mathcal{T}_j = \{2^i \mid i \in \mathbb{N}_0,\, 2^i \leq T_j\} \cup \{T_j\}, \tag{9}$$

i.e., all powers of two up to T_j, plus the final token position. The stacked activation tensor for input j at layer l is:

$$\mathbf{A}_l^{(j)} = \bigoplus_{t \in \mathcal{T}_j} a_{l,t}^{(j)} \in \mathbb{R}^{|\mathcal{T}_j| \times d_l}, \tag{10}$$

where $\bigoplus$ denotes stacking along the sample axis. The full *activation dataset* is then $\mathcal{A} = \{\mathbf{A}_l\}_{l \in \mathcal{L}}$, with:

$$\mathbf{A}_l = \left[\mathbf{A}_l^{(1)}; \ldots; \mathbf{A}_l^{(N)}\right] \in \mathbb{R}^{K \times d_l} \tag{11}$$

yielding $K = \sum_{j=1}^{N} |\mathcal{T}_j| > N$ total activation samples.

A.2 Activation Classifier (AC)

The Activation Classifier (AC) architecture is detailed in Algorithm 2.

The activation encoder $\mathcal{C}_{\text{enc}}$ is an m-layer MLP, with each layer applying a LayerNorm, linear projection, and ReLU nonlinearity:

$$\text{Linear}_i : \mathbb{R}^{h_{i-1}} \to \mathbb{R}^{h_i}, \quad h_0 = D$$
$$\mathbf{x}^{(i)} = \text{ReLU}\Big(\text{Linear}_i\Big(\text{LN}(\mathbf{x}^{(i-1)})\Big)\Big)$$

producing a latent representation $\mathbf{z} = \mathbf{x}^{(m)} \in \mathbb{R}^{h_m}$ per activation sample.

The classification head $\mathcal{C}_{\text{head}}$ is composed of a mean-pooling aggregation over all K encoded samples followed by a linear projection and a sigmoid:

$$\bar{\mathbf{z}} = \frac{1}{K} \sum_{i=1}^{K} \mathbf{z}_i \in \mathbb{R}^{h_m}$$
$$\hat{y} = \sigma(\text{Linear}_{h_m \to 1}(\bar{\mathbf{z}})) \in [0, 1]$$

where σ is the Sigmoid function and $\hat{y}$ is the predicted probability that the target model is backdoored.

A.3 Activation Transformer (AcT)

The Activation Transformer (AcT) architecture is detailed in Algorithm 3. In general, the activation dimension d_l may vary across layers of the same model (e.g., when capturing activations from non-residual-stream locations), which would require more careful handling and processing of activation samples. However, in the residual stream of transformer-based LLMs, hidden states are passed through residual connections at a fixed dimension throughout the network, meaning $d_l = d$ is constant across all captured layers. This is the case for all transformer-based architectures, and specifically the ones we consider. We therefore present and implement the algorithm under this assumption, which simplifies the construction of 2D positional encodings and allows activations from all layers to be stored and reshaped into a single unified tensor for processing, as detailed in Algorithm 3. Note that this assumption concerns layers within a single model, and is orthogonal to the cross-architecture challenge addressed by AcT, where d may differ across different model architectures (e.g., $d = 4096$ for Llama-2-7B vs. $d = 5120$ for Llama-2-13B), which AcT handles via its block-based decomposition.

Algorithm 1 BackdoorID

Require: Target model $\mathcal{M}$, classifier $\mathcal{C}$, target layers $\mathcal{L}$, number of inputs N, generation args κ

Input Generation:

$\mathcal{V} \leftarrow \mathcal{M}.\texttt{vocab} \setminus \mathcal{V}_{\text{special}}$ ▷ Vocabulary excluding special tokens

$\mathcal{D}_{\text{in}}^{\mathcal{M}} \leftarrow \left\{\mathcal{M}.\texttt{generate}(\tilde{x}_k;\, \kappa) \;\middle|\; \tilde{x}_k \sim \mathcal{U}(\mathcal{V})\right\}_{k=1}^{N}$ ▷ Autoregressively generate N inputs from randomly sampled seed tokens (Eq. 3)

Activation Extraction:

$H \leftarrow \mathcal{M}(\mathcal{D}_{\text{in}}^{\mathcal{M}})$ ▷ Forward pass all inputs; Capture hidden states of each from target layers $\mathcal{L}$

$\mathcal{A} \leftarrow \texttt{Extract}(H,\, \mathcal{L})$ ▷ $\mathcal{A} = \{A_1, \ldots, A_{|\mathcal{L}|}\}$: extracting activation samples from target layers via standard or dynamic strategy

Encoding:

$Z \leftarrow \mathcal{C}_{\text{enc}}(\mathcal{A})$ ▷ Encode all K activation samples: $Z = [z_1;\, \ldots;\, z_K] \in \mathbb{R}^{K \times d_{\text{enc}}}$

Classification:

$\hat{y} \leftarrow \mathcal{C}_{\text{head}}(Z)$ ▷ Aggregate encodings via $\mathcal{C}_{\text{head}}$ to get single prediction $\hat{y} \in [0, 1]$

return $\hat{y}$

Algorithm 2 Activation Classifier (AC) Architecture

Require: hyper-parameters: Hidden dimensions $[h_1, \ldots, h_m]$

Activation Encoder $\mathcal{C}_{\text{enc}}$:

Require: Input: Activation dataset $\mathcal{A} = \{l : \mathbf{A}_l\}_{l \in \mathcal{L}}, \quad \mathbf{A}_l \in \mathbb{R}^{K \times d_l}$

$\mathbf{x} \leftarrow \text{Concat}([\mathbf{A}_{l_1},\, \ldots,\, \mathbf{A}_{l_{|\mathcal{L}|}}],\ \text{dim}{=}{-1}) \in \mathbb{R}^{K \times D}$ ▷ Concatenate activations of all layers within each sample, $D = \sum_{l \in \mathcal{L}} d_l$

$\mathbf{Z} \leftarrow \mathcal{C}_{\text{enc}}(\mathbf{x}) \in \mathbb{R}^{K \times h_m}$ ▷ Encode each activation sample.

return Z

Classification Head $\mathcal{C}_{\text{head}}$:

Require: $\mathbf{Z} \in \mathbb{R}^{K \times h_m}$ ▷ All K activation sample encodings

$\hat{y} \leftarrow \mathcal{C}_{\text{head}}(Z) \in [0, 1]$ ▷ Aggregate all encodings into a single scalar

return $\hat{y}$ ▷ predicted probability: clean vs. backdoored

Block Size Constraint. If for some layer $l \in \mathcal{L}$ the activation dimension d_l is not divisible by the block size d_{block}, we define

$$d_l' = d_{\text{block}} \cdot \left\lfloor \frac{d_l}{d_{\text{block}}} \right\rfloor$$

and only partition the first d_l' coordinates into blocks of size d_{block}, discarding the remaining $d_l - d_l'$ coordinates.

A.4 Training Details

Training The classifier is optimized with AdamW using binary cross-entropy (BCE) loss defined as:

$$\ell_i = y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

$$\mathcal{L}_{\text{BCE}} = \frac{1}{N} \sum_{i=1}^{N} -\ell_i \tag{12}$$

where $y_i \in \{0, 1\}$ and $\hat{y}_i \in [0, 1]$ are the label and classifier prediction of sample i respectively.

Activation Caching. To reduce computational overhead during training, the activation set $\mathcal{A}_i$ for each model M_i is cached after the first pass over the target model. In subsequent epochs, cached activations are reused directly, eliminating redundant input generations and forward passes through the target models, significantly reducing training time. This caching is transparent to the classifier and did not affect classifier performance in our preliminary testing.

B Appendix - Experimental Setup Details

B.1 Benchmarks

Here we present a more detailed overview of the benchmarks used in our evaluations.

PADBench. (Sun et al., 2025) The benchmark consists of adapters, encompassing both benign and backdoored versions. It spans several LLMs, from which we use Llama-2 (7B, 13B), Llama-3 (8B), Qwen1.5 (7B).

The benchmark incorporates a wide array of tasks and attack methodologies to evaluate detection performance. Due to computational limitations, we focus on specific subsets of the benchmark.

Datasets: PADBench includes task-specific datasets such as IMDB (sentiment classification), AG News (topic classification), and SQuAD (question answering), as well as Instruction-Following (IF) datasets like toxic_backdoors_alpaca (TBA) and toxic_backdoors_hard (TBH).

Both IMDB and AG News are classification tasks with two and four classes respectively, adapted to the generative setting by prompting the model to generate the class label as text. The target of the attacks in these datasets is the generation of a specific class label when the trigger is present. SQuAD is a question-answering dataset where the model is expected to generate accurate answers to given questions. In this case, the backdoor attacks aim to manipulate the model into producing a predetermined toxic answer when the trigger is present. Lastly, TBA and TBH are datasets based on the instruction-following dataset alpaca, and poisoned with InsertSent-style attacks, which we refer to as Sentence, to induce different toxic responses in presence of the trigger. TBH uses different sentences related to Manchester United as trigger, rather than fixed trigger as in TBA.

Attack Types: The benchmark features four primary textual backdoor attacks:

- *InsertSent (Dai et al., 2019):* Inserts a fixed trigger sentence into the input at random locations.
- *RIPPLES (Kurita et al., 2020):* Injects rare trigger words (e.g., "cf", "mn") as a poisoning strategy.
- *Syntactic (Qi et al., 2021b):* Uses specific syntactic templates as triggers, modifying sentence structures.
- *StyleBkd (Qi et al., 2021a):* Leverages text style transfer (e.g., biblical style) as the backdoor trigger.

BAIT-ModelZoo. (Shen et al., 2025) The benchmark consists of 91 models, encompassing both benign and backdoored versions. It spans several LLM architectures, including the LLaMA-2 (7B), LLaMA-3 (8B), Mistral (7B). BAIT-ModelZoo can be split into several subsets, each a collection of models trained for a specific combination of architecture and dataset. Each subset contains multiple models trained with different choices of trigger and target.

Datasets: BAIT-ModelZoo focuses on Instruction-Following (IF) datasets Alpaca and Self-Instruct. In this benchmark, backdoor attacks aim to manipulate the model into producing *fixed* outputs when the trigger is present (i.e. given the trigger is present in the input, the output will be the same regardless of the input content).

Attack Types: The benchmark features a single textual backdoor attack, *CBA (Huang et al., 2024):* Composite Backdoor Attack, which is composed of multiple triggers. The backdoor behavior is activated only when all triggers are present in the input.

B.2 Implementation Details.

BackdoorID Framework. To train both classifiers we used learning rate of $3e^{-4}$ and weight decay of $1e^{-6}$ and utilized the caching mechanism described in Appendix A.

AC Architecture. For the AC the input dimension, h_0 was dictated by the size of collected model activations. However, the hidden dimensions were kept the same across all experiments. We used two hidden dimensions $h_1 = 512, h_2 = 256$, for a total of $m = 3$ layers with the final output size of 1.

Hyperparameter	Value
Block size (d_{block})	256
Encoding dimension (d_{enc})	64
Transformer layers (n_{layers})	1
Attention heads (n_{head})	4
FFN dimension (d_{ff})	256
Dropout (p)	0.2
Normalization	LayerNorm

Table 3: AcT architecture hyper-parameters.

AcT Architecture. In table 3 we describe the architectural hyper-parameters used for the AcT classifier across all experiments. We used the same parameters for Transformer Encoders of both the activation encoder and classification head (which

includes any parameter below the horizontal line, starting with the "Transformer layers (n_{layers})" parameter). We utilized the implementation of the PyTorch transformer encoder, which follows Vaswani et al. (2017), using full bi-directional attention across all inputs of each sequence.

Hardware. All experiments were run on machines with Nvidia L40 or 6000RTX PRO GPUs.

B.3 RQ1

For each parameter configuration, accuracy is averaged over 3 runs with different random seeds.

Experiment 1: Effect of Input Length. Both the standard and dynamic activation extraction strategies are evaluated across the IMDB, AGNews, and TBH subsets of PADBench. To ensure a fair comparison across strategies, we fix the target number of activation samples at $K = 64$ and adjust the number of generated inputs N accordingly. For the dynamic strategy, the number of activation samples per input depends on T, and $K = 64$ may not be exactly achievable for all values of T. In such cases, N is chosen as the largest integer such that the resulting number of activation samples does not exceed 64. For example, at $T = 4$ the dynamic strategy yields 3 activation samples per input, since 64 is not divisible by 3, we set $N = 21$, resulting in $K = 63$.

Experiment 2: Effect of Sample Count. All experiments use the TBH, Llama-2 7B subset. For each existing dataset input source, N inputs are sampled, matching the count used for generation. Alpaca serves as an in-distribution reference, as it is the dataset used to poison the model in the TBH subset. GSM8K and AG News, a mathematics dataset and news articles corpora, serve as an out-of-distribution reference, being distinct from the instruction-following domain used to implant the backdoor. For generated inputs we use $T = 8$, while for existing datasets we fix $T = 64$ to preserve input context. We validate that this choice does not significantly affect classifier accuracy in Appendix D.1. Since at $T = 8$ the dynamic strategy yields exactly 4 activation samples per input (at positions $1, 2, 4, 8$), we set $N = K/4$ for dynamic extraction, ensuring the total number of activation samples remains exactly K across both strategies.

B.4 RQ2

B.4.1 BAIT

To run BAIT (Shen et al., 2025), we used the publicly available code. At the time of our experiments, the released BAIT code had been updated relative to the algorithm described in the original paper. Most notably, a judge LLM was added to assess candidate targets. Hyperparameters are listed in Table 4, specifically, `batch_size` and `min_target_len` were utilized to improve runtime. Default values were used for any unspecified parameter.

B.4.2 PEFTGuard

Parameter-Efficient Fine-Tuning. PEFT methods adapt a pretrained model M_θ to a downstream task by optimizing only a small set of auxiliary parameters Δ, where $|\Delta| \ll |\theta|$, while keeping the base parameters θ frozen. The resulting parameters Δ, commonly referred to as *adapters*, are incorporated into the base model via a method-specific operation $\oplus$, yielding the adapted model $M_{\theta\oplus\Delta}$. Due to the compact size of Δ, PEFT adapters are lightweight, easy to distribute, and have a popular method for sharing fine-tuned LLMs via public model repositories.

Implementation. PEFTGuard (Sun et al., 2025) operates directly on adapter parameters Δ without requiring access to the base model. Although adapters are typically stored in low-rank factorized form (e.g., $\Delta = BA$ for LoRA (Hu et al., 2022) with rank $r \ll d$), PEFTGuard first materializes each adapter weight matrix to its full dimensionality by computing $\Delta^{(l)}_{\text{full}} = 0 \oplus \Delta^{(l)}$, where $0 \in \mathbb{R}^{d\times k}$ represents a zero matrix and $\oplus$ denotes the adapter merging operation of the PEFT method (e.g., for LoRA: $\theta\oplus\Delta = \theta + BA$). The materialized adapter weights from all layers are then transformed into a fixed-size tensor and fed into a meta-classifier that outputs a binary prediction indicating whether the adapter is clean or backdoored.

Adaptation to Full Models (PEFTGuard$_{\text{FM}}$). To extend PEFTGuard to full-model detection, we adapt the method by modifying only the input representation while keeping all other components identical. Specifically, instead of materializing adapters in isolation via $0 \oplus \Delta^{(l)}$, we compute the full merged model weights $\theta'^{(l)} = \theta^{(l)} \oplus \Delta^{(l)}$, where $\theta^{(l)} \in \mathbb{R}^{d\times k}$ represents the base model weights at layer l. Critically, since both $0\oplus\Delta^{(l)}$ and $\theta^{(l)}\oplus\Delta^{(l)}$ yield tensors of identical dimensionality $\mathbb{R}^{d\times k}$, the

Subset \ Benchmark	Judge Model	batch_size	min_target_len	input dataset
(TBH, Llama3_8B, Sentence)	Qwen3 14B	50	4	Alpaca
(TBH, Qwen1_5 7B, Sentence)			4	Alpaca
(TBH, Llama2 7B, Sentence)		80	4	Alpaca
(TBH, Llama2 13B, Sentence)			4	Alpaca
(SQuAD, Llama2 7B, InsertSent)			1	squadv2
(IMDB, Llama2 7B, All)			1	IMDB
(AG News, Llama2 7B, All)			1	AG News
BAIT-ModelZoo		100	4	Alpaca Self-Instruct

Table 4: Parameters for BAIT experiments. subsets are represented by tuple of respective dataset, model, and attack.

subsequent feature transformation, classifier architecture, training procedure, and all hyperparameters remain unchanged from the original PEFTGuard. This enables $\text{PEFTGuard}_{\text{FM}}$ to detect backdoors in full models (base + adapter) using the same framework designed for adapter-only detection.

Parameters. We used the code published by the authors on GitHub, supplementing missing functionalities with our own implementations. The correctness of these implementations is supported by results comparable to those reported in the paper (see Table 6).

Training was conducted with the maximum feasible batch size of 1, limited by GPU memory constraints, a learning rate of 1×10^{-4}, and a weight decay of 1×10^{-5}. Since no classifier architecture was available for Mistral 7B, we applied the Llama 3 8B classifier, given the highly similar architectures of the two models.

B.4.3 BackdoorID (Ours)

We evaluate BackdoorID using dynamic extraction, a maximum input length of $T = 2$ tokens (determined as the best overall across subsets), and $K = 64$ activation samples. All other hyperparameters follow the settings described in B.2.

B.4.4 Additional Experiments

We present two additional sets of experiments in Tables 5 and 6.

First, we evaluate PEFTGuard in its original, adapter-only setting. In contrast to the full-model variant described in Section B.4.2, both variants receive weights from the same self-attention layers as input, specifically, the query and value projection matrices, but differ in which weights those layers correspond to. $\text{PEFTGuard}_{\text{FM}}$ passes the weights of the resulting full model (base + adapter), whereas PEFTGuard passes only the adapter weights for those layers, following the original paper (Sun et al., 2025). This distinction is important because PEFTGuard was originally designed and validated in the adapter-only regime, where the backdoor is assumed to be isolated within the adapter parameters. Evaluating in this setting allows us to first reproduce the results reported in the original paper and verify the correctness of implementation before applying the method in our more challenging full-model setting. All other hyper-parameters remain identical to those described in Section B.4.2, and we evaluate on all subsets used in RQ2.

Second, we evaluate BackdoorID using the standard activation extraction strategy, complementing the dynamic extraction results already reported in RQ2. Recall that the standard strategy captures only the last-token activation of each generated input, yielding exactly one activation sample per input (Section A.1.1). We set $N = 64$, so that $K = 64$, matching the sample count used with dynamic extraction. Keeping K fixed across both strategies enables a direct comparison of their detection performance. We evaluate both the AC and AcT classifiers under the standard extraction strategy across all subsets in Table 5 .

B.5 RQ3

To evaluate cross-architecture generalization, we train and evaluate the AcT classifier on a combined dataset spanning three TBH subsets: Llama-2 7B, Llama-2 13B, and Qwen 1.5 7B. These subsets cover three distinct model architectures with two different activation dimensionalities: Llama-2 7B and Qwen 1.5 7B share the same activation dimensionality (4096), while Llama-2 13B differs (5120).

Dataset Construction. The combined dataset is formed by merging the corresponding splits of each individual subset. Specifically, the training set is the union of the three subsets' training splits, and similarly for validation and test. The per-subset splits follow the same 1:1:1 adapter-level partition-

Benchmark	Subset			PEFTGuard$_{FM}$	BAIT	BackdoorID (ours) (standard)		BackdoorID (ours) (dynamic)	
	Dataset	Architecture	Attack			AC	AcT	AC	AcT
PADBench	IMDB	Llama2 7B	InsertSent	50%	50%	95.3%	94%	**96%**	**96%**
	IMDB	Llama2 7B	StyleBkd	50%	50%	84%	**88.7%**	84.7%	82%
	IMDB	Llama2 7B	RIPPLE	50%	46%	**100%**	**100%**	**100%**	99.3%
	IMDB	Llama2 7B	Syntactic	50%	56%	74%	56.7%	**74.7%**	49.3%
	AG News	Llama2 7B	InsertSent	59.3%	50%	93.3%	84.7%	**94%**	90.7%
	AG News	Llama2 7B	StyleBkd	60.7%	52%	**100%**	98.7%	99.3%	98%
	AG News	Llama2 7B	RIPPLE	55.3%	50%	93.3%	**94%**	93.3%	93.3%
	AG News	Llama2 7B	Syntactic	51.3%	46%	**98%**	91.3%	96%	89%
	SQuAD	Llama2 7B	InsertSent	50%	84%	**100%**	**100%**	**100%**	**100%**
	TBH	Llama2 7B	Sentence	87.3%	92%	93.3%	**96%**	93.3%	92%
	TBH	Llama2 13B	Sentence	67.3%	84%	80%	86.7%	84.7%	**87.3%**
	TBH	Llama3 8B	Sentence	68%	48%	94%	94.7%	94.7%	96%
	TBH	Qwen1.5 7B	Sentence	53.3%	82%	**100%**	99.3%	**100%**	99.3%
	TBA	Llama2 7B	Sentence	62%	62%	**96.7%**	94.7%	96%	93.3%
BAIT ModelZoo	SI + Alpaca	Llama2 7B	CBA	77.7%	**100%**	**100%**	**100%**	**100%**	**100%**
	SI + Alpaca	Llama3 8B	CBA	63.9%	91.7%	**100%**	**100%**	**100%**	**100%**
	SI + Alpaca	Mistral 7B	CBA	91.7%	**100%**	**100%**	72.2%	**100%**	83.3%

Table 5: Evaluation of BackdoorID with baselines on diverse set of benchmark subsets.

ing used across all other experiments, and class balance is maintained within each split. Each subset contains 150 adapters, yielding a combined dataset of 450 models in total, with approximately 150 models in each of the training, validation, and test splits.

Classifier Training. The AcT classifier is trained jointly on the combined training set, with no architecture-specific components or separate per-architecture classifiers. This directly evaluates the model's ability to learn a shared backdoor detection signal across heterogeneous architectures and activation dimensionalities. Training uses a high patience value for early stopping, and as in the rest of the experiment, the checkpoint corresponding to the lowest validation loss is selected for evaluation.

Hyper-parameters. We use the dynamic extraction strategy with $K = 64$ activation samples per model, using self-generated inputs. We set the maximum input length to $T = 8$ tokens, motivated by the RQ1 analysis, which identified it as the optimal input length for the (TBH, Llama2 7B) subset.

C Appendix - Results

C.1 RQ2: Detection Performance

Table 2 reports detection accuracy across a wide range of benchmark subsets.

Table 5 also includes results for BackdoorID under the standard extraction strategy with K=64, alongside the dynamic extraction results from Table 2. The two strategies yield broadly comparable detection accuracy across subsets. This is consistent with the RQ1 findings (Figure 2, left), which show that at short input lengths such as T=2, the performance gap between standard and dynamic extraction is small.

Table 6 reports results for PEFTGuard$_{FM}$ and PEFTGuard (adapter-only), which serve primarily to validate and reproduce the original results of Sun et al. (2025). Indeed, PEFTGuard achieves very high detection accuracy across most evaluated subsets, consistent with the results reported in the original paper, confirming the correctness of implementation. We note that our reproduced numbers are lower than originally reported, which is consistent with the reported sensitivity of PEFTGuard to training set size in Sun et al. (2025), which demonstrated decreasing performance under reduced training data, specifically below 100 training samples, while we use 50.

In stark contrast, and despite identical input dimensionality, classifier architecture and hyper-parameters, PEFTGuard (full-model) generally achieves much lower accuracy, and with high variance across subsets, ranging from random-guessing performance of (50%) on several PADBench subsets to above 90% on the BAIT-ModelZoo Mistral-7B subset. We hypothesize that merging adapter parameters into the base model diffuses the backdoor signal, making full-model detection inherently more challenging for weight-based classifiers,

Benchmark	Subset			PEFTGuard	PEFTGuard$_{FM}$
	Dataset	Architecture	Attack		
PADBench	IMDB	Llama2 7B	InsertSent	**99.3%**	50%
	IMDB	Llama2 7B	StyleBkd	**88%**	50%
	IMDB	Llama2 7B	RIPPLE	**100%**	50%
	IMDB	Llama2 7B	Syntactic	**82.7%**	50%
	AG News	Llama2 7B	InsertSent	**98%**	59.3%
	AG News	Llama2 7B	StyleBkd	**100%**	60.7%
	AG News	Llama2 7B	RIPPLE	**98.7%**	55.3%
	AG News	Llama2 7B	Syntactic	**100%**	51.3%
	SQuAD	Llama2 7B	InsertSent	**100%**	50%
	TBH	Llama2 7B	Sentence	**98%**	87.3%
	TBH	Llama2 13B	Sentence	**92%**	67.3%
	TBH	Llama3 8B	Sentence	**100%**	68%
	TBH	Qwen1.5 7B	Sentence	**99.3%**	53.3%
	TBA	Llama2 7B	Sentence	**98.7%**	62%
BAIT ModelZoo	SI + Alpaca	Llama2 7B	CBA	**100%**	77.7%
	SI + Alpaca	Llama3 8B	CBA	**94.5%**	63.9%
	SI + Alpaca	Mistral 7B	CBA	88.9%	**91.7%**

Table 6: Comparison of PEFTGuard (adapter only) with PEFTGuard$_{FM}$ (full-model) on all evaluated subsets.

though more research is needed to confirm this hypothesis.

D Additional Results

D.1 Optimal Length for Inputs from Existing Datasets

In Section B.3, we use $T = 64$ when using inputs from existing datasets, and note that this choice does not significantly affect classifier accuracy. Here we present the ablation experiment supporting this claim.

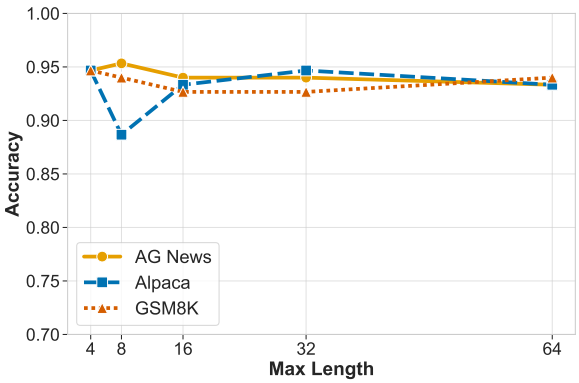

Figure 4: BackdoorID[AC] accuracy as function of input length, using inputs from existing datasets.

Experiment Setup. We train the AC classifier using the standard activation extraction strategy on inputs drawn from three existing input datasets, namely Alpaca, AG News, and GSM8K, on the (TBH, Llama-2-7B) subset. We vary the input length $T \in \{4, 8, 16, 32, 64\}$ and report the resulting classifier accuracy in Figure 4. Alpaca serves as an in-distribution reference, while AG News and GSM8K serve as out-of-distribution references. Following the same protocol as in RQ1, reported results are average over 3 runs for each configuration.

Results. As shown in Figure 4, detection accuracy remains approximately constant across all input length values, T, for all three input sources, consistently around $\sim$94%. This stands in sharp contrast to the generated-input setting, where accuracy is sensitive to T and eventually degrades as inputs grow longer (Figure 2, left). The insensitivity to T observed here confirms that classifier performance is robust to the choice of input length when using inputs from existing datasets. Our selection of $T = 64$ is therefore a well-motivated and empirically supported choice, as it preserves the natural semantic context of real-world inputs without incurring any accuracy penalty.

Algorithm 3 Activation Transformer (AcT) Architecture

Require: hyper-parameters: Block size d_{block}, model dim d_{enc}, heads n_{head}, encoder layers $n^{(1)}_{\text{layers}}$, classification layers $n^{(2)}_{\text{layers}}$, dropout p

---

Activation Encoder $\mathcal{C}_{\text{enc}}$:

Require: Input: Activation dict $\mathcal{A} = \{l : \mathbf{A}_l\}_{l \in \mathcal{L}}, \quad \mathbf{A}_l \in \mathbb{R}^{K \times d}$

Activation Aggregation:

$\mathbf{X} \leftarrow \text{Stack}([\mathbf{A}_1, \ldots, \mathbf{A}_{|\mathcal{L}|}], \text{ dim=1})$ ▷ $\mathbf{X} \in \mathbb{R}^{N \times |\mathcal{L}| \times d}$

Block Partitioning:

$M \leftarrow d/d_{\text{block}}$ ▷ Number of blocks per layer; requires $d \equiv 0 \pmod{d_{\text{block}}}$

$\mathbf{X}_{\text{blk}} \leftarrow \text{Reshape}(\mathbf{X}, (K, |\mathcal{L}|, M, d_{\text{block}}))$

Input Projection:

$\mathbf{H} \leftarrow \text{Linear}_{d_{\text{block}} \to d_{\text{enc}}}\big(\text{LayerNorm}(\mathbf{X}_{\text{blk}})\big)$ ▷ $\mathbf{H} \in \mathbb{R}^{N \times |\mathcal{L}| \times M \times d_{\text{enc}}}$

Factored 2D Positional Encoding:

$\mathbf{H}[\ldots, : d'] \mathrel{+}= \mathbf{P}_{\text{block}}[\ldots, : M, :]$ ▷ Block position, upper half; $d' = d_{\text{enc}}/2$

$\mathbf{H}[\ldots, d' :] \mathrel{+}= \mathbf{P}_{\text{layer}}[\ldots, : |\mathcal{L}|, :, :]$ ▷ Layer identity, lower half

where $\mathbf{P}_{\text{block}} \in \mathbb{R}^{1 \times 1 \times M_{\text{max}} \times d'}$, $\mathbf{P}_{\text{layer}} \in \mathbb{R}^{1 \times L_{\text{max}} \times 1 \times d'}$ are learnable

Sequence Construction:

$\mathbf{H}_0 \leftarrow \text{Reshape}(\mathbf{H}, (K, |\mathcal{L}| \cdot M, d_{\text{enc}}))$ ▷ Flatten layers $\times$ blocks into sequence

$\mathbf{H}_0 \leftarrow [\mathbf{t}^{(1)}_{\text{CLS}}; \mathbf{H}_0]$ ▷ Prepend learnable CLS token; $\mathbf{H}_0 \in \mathbb{R}^{K \times (|\mathcal{L}| \cdot M + 1) \times d_{\text{enc}}}$

Transformer Encoding (per activation sample):

$\mathbf{H}_{n^{(1)}_{\text{layers}}} \leftarrow \text{TransformerEncoder}(\mathbf{H}_0)$ ▷ Pre-norm, multi-head self-attention + FFN

$\mathbf{Z} \leftarrow \mathbf{H}_{n^{(1)}_{\text{layers}}}[:, 0, :]$ ▷ CLS output serves as encoding of each sample; $Z \in \mathbb{R}^{K \times d_{\text{enc}}}$

return $\mathbf{Z}$

Classification Head $\mathcal{C}_{\text{head}}$:

Require: Per-sample encodings $\mathbf{Z} \in \mathbb{R}^{K \times d_{\text{enc}}}$

Cross-Sample Aggregation:

$\mathbf{Z}_0 \leftarrow [\mathbf{t}^{(2)}_{\text{CLS}}; \mathbf{Z}].\text{unsqueeze}(0)$ ▷ Prepend second learnable CLS token; $\mathbf{Z}' \in \mathbb{R}^{1 \times (K+1) \times d_{\text{enc}}}$

$\mathbf{Z}_{n^{(2)}_{\text{layers}}} \leftarrow \text{TransformerEncoder}(\mathbf{Z}_0)$ ▷ Attend across all K activation sample encodings

$\mathbf{h}_{\text{CLS}} \leftarrow \mathbf{Z}_{n^{(2)}_{\text{layers}}}[0, 0, :]$ ▷ $\mathbf{h}_{\text{CLS}} \in \mathbb{R}^{d_{\text{enc}}}$

Prediction:

$y \leftarrow \text{Linear}_{d_{\text{enc}} \to 1}\big(\text{LayerNorm}(\mathbf{h}_{\text{CLS}})\big) \in \mathbb{R}$

$\hat{y} \leftarrow \text{Sigmoid}(y) \in [0, 1]$

return $\hat{y}$ ▷ predicted probability: clean vs. backdoored