Architectural Backdoors in Deep Learning: A Survey of Vulnerabilities, Detection, and Defense

VICTORIA CHILDRESS, JOSH COLLYER, and JODIE KNAPP, The Alan Turing Institute, UK

Architectural backdoors pose an under-examined but critical threat to deep neural networks, embedding malicious logic directly into a model's computational graph. Unlike traditional data poisoning or parameter manipulation, architectural backdoors evade standard mitigation techniques and persist even after clean retraining. This survey systematically consolidates research on architectural backdoors, spanning compiler-level manipulations, tainted AutoML pipelines, and supply-chain vulnerabilities. We assess emerging detection and defense strategies, including static graph inspection, dynamic fuzzing, and partial formal verification, and highlight their limitations against distributed or stealth triggers. Despite recent progress, scalable and practical defenses remain elusive. We conclude by outlining open challenges and proposing directions for strengthening supply-chain security, cryptographic model attestations, and next-generation benchmarks. This survey aims to guide future research toward comprehensive defenses against structural backdoor threats in deep learning systems.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Neural networks**; • **Security and privacy** → **Systems security**; **Software security engineering**; **Intrusion/anomaly detection and malware mitigation**.

Additional Key Words and Phrases: Architectural backdoors, Neural-network security, Compiler Trojans, Supply-chain assurance, Formal verification, AutoML vulnerabilities

1 Introduction

Architectural backdoors pose a persistent and under-examined threat to deep neural networks. By embedding malicious logic directly into the computational graph, they evade traditional defenses and persist even after retraining. Because the exploit is hard-wired (e.g., as an extra branch, gating layer, or rerouted edge), it typically remains dormant and can persist after weight reinitialization. These structures only activate in response to attacker-defined inputs or trigger patterns. By encoding the trigger logic directly in new or rewired operators, architectural backdoors are immune to standard mitigation techniques like data cleansing, weight resets, or fine-tuning alone. Conceptually, such a structural exploit is akin to a hardware Trojan gate hidden in an integrated circuit: the malicious sub-graph lies dormant until a secret trigger activates it [69, 75]. Fine-grained pruning can disable simple single-path backdoors [46], but sophisticated or distributed sub-graphs still evade such defenses [51]. Prior backdoor surveys have largely focused on data or weights; by contrast, we illuminate architectural backdoors as an emerging threat class requiring dedicated study. Earlier reviews focus on data- or weight-level Trojans [3, 41, 87]. Instead, we will synthesize the emerging literature on architectural backdoors and their unique detection and mitigation challenges. Recent work has revealed that architectural backdoors can do more than trigger misclassifications: by exploiting within-batch inference they can leak or manipulate the outputs of other users who share the same batch, breaking the isolation guarantees of large-scale model serving [36]. This privacy-critical scenario extends the threat surface first highlighted by Bober-Irizar et al. [6], who coined the term Model Architectural Backdoor (MAB) for malicious logic wired directly into a network's computational graph.

Structural backdoors resist standard removal: once a malicious sub-graph is embedded in the architecture, it survives weight re-initialization and clean retraining, leaving safety-critical models exposed. Automated machine learning (AutoML) and neural architecture search (NAS) pipelines hide such logic inside increasingly complex topologies, making detection harder. Real-world evidence now exists: *AI*'s *Guardian* scanner has examined 4.47 million model versions in 1.41 million Hugging

Table 1. **Common backdoor types**

Type	Description
Data poisoning	Injecting triggers into training samples (often mislabeled) to activate hidden misbehavior at inference
Weight backdoor	Manipulating model parameters or fine-tuning with malicious objectives to cause trigger-based misclassifications
Architectural	Modifying the network's structure (e.g., extra subgraphs, backdoor layers) that persist even under clean retraining

Face repositories and has flagged 352,000 unsafe or suspicious issues across 51,700 distinct models, including the popular *SoccerTwos* ONNX file, for architectural-backdoor–like sub-graphs [2]. Recent work such as HiddenLayer's *Shadow Logic* likewise shows trigger-activated misbehavior persisting even after full ImageNet retraining [29]. In a safety-critical setting, embedded triggers might remain dormant until activated by carefully crafted stimuli, leading to catastrophic failures at the worst possible moments. The combination of stealth and persistence thus demands dedicated study of architectural backdoors, filling a gap left by previous surveys on data- and weight-based backdoors.

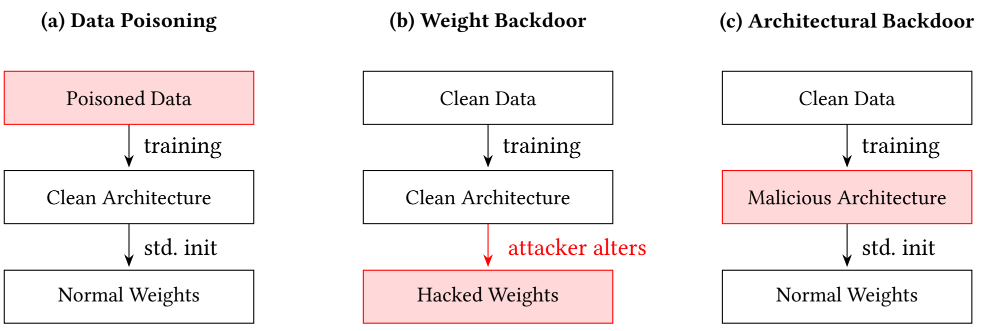

Fig. 1. Three main backdoor insertion points. (a) data poisoning, (b) weight backdoor, and (c) architectural manipulation. Attacker-altered elements are shown in red.

1.1 Contributions and Organization

Our main contributions are:

- **Comprehensive Taxonomy of Architectural Backdoors.** We categorize different architectural attacks, including sub-network modifications, compiler-level backdoor insertion, and AutoML vulnerabilities, detailing their distinct threat models and stealth properties.
- **Detection and Mitigation Strategies.** We survey existing techniques (e.g., static graph inspection, dynamic fuzzing, formal verification, subgraph pruning) and assess their efficacy against architecture-centric backdoors.
- **Open Challenges and Future Directions.** We highlight research gaps such as supply-chain security, robust model verification at scale, and multi-branch triggers, emphasizing the need for comprehensive defenses that go beyond data or weight checks.

§2 reviews classical backdoor mechanisms and clarifies how architectural attacks fit within the broader landscape. §3 presents our taxonomy of architectural backdoors. §4 delves into detection techniques, followed by mitigation and model repair strategies in **§5**. **§6** surveys available benchmarks, datasets, and empirical evaluations for assessing architectural backdoors, and **§7** discusses open challenges and future directions. Finally, **§8** concludes with implications for securing AI models against architectural backdoors.

2 Background and Related Work

A backdoor can be implanted (i) in the training data, (ii) in the model weights, or (iii) in the network architecture. The remainder of this section explains why architectural backdoors demand separate treatment and motivates the taxonomy that follows.

2.1 Classical vs. Architectural Backdoors

Classical (Data- and Weight-Based) backdoors. Most early backdoor research, as surveyed in prior reviews [3, 41, 44, 87], centered on manipulating either training data or model weights. A prototypical data-poisoning approach is *BadNets* [24], which injects small patterns into a subset of training samples, causing misclassification only when the trigger is present. Meanwhile, weight-based attacks [48] alter parameters (e.g., via fine-tuning or partial re-initialization) to produce attacker-chosen outputs upon detecting a secret trigger [38, 39]. These methods may have minimal footprints in weights or data, making them difficult to prune. Recent work by Goldwasser *et al.* [23] demonstrated the existence of undetectable weight-edited backdoors, where adversarial parameter edits are computationally indistinguishable from clean weights. These attacks blur the line between whitebox and blackbox threat models, as even full model access may not enable reliable detection. Classical backdoor defenses often focus on spotting anomalous activations or triggers (e.g., neuron pruning, trigger inversion), which can be partially effective if the backdoor is embedded in data or parameters. However, such methods largely assume suspect training samples or suspicious weights can be identified and removed; that assumption fails if the malicious logic is baked into the graph itself.

Architectural backdoors. Recent works [6, 13] show that structural modifications survive weight re-initialization, retraining on clean data, or standard pruning strategies. A controlled study by Langford *et al.* [37] demonstrates this empirically: even after full weight re-initialization followed by clean *ImageNet* retraining, their architectural Trojan preserved a 96.2% attack-success rate, whereas a baseline *BadNets* weight backdoor fell below 2%. Earlier proof-of-concept work by Qi *et al.* demonstrated that a tiny malicious sub-network can even be inserted after training, at deployment time, yet still pass unnoticed and retain 100 % attack success once triggered [62]. This persistence arises because the malicious branch is weight-agnostic: its routing or gating logic is hard-wired into the graph topology rather than stored in learned parameters. Re-initializing or retraining the weights leaves those fixed routes untouched, so the backdoor re-emerges as soon as training converges. By wiring malicious routing or gating logic directly into the computational graph, attackers ensure it can be selectively activated by triggers. Because this backdoor is not encoded in the parameters or the data, standard defenses (data cleansing, weight resets) cannot remove the malicious subgraph. Figure 2 offers a schematic example of a hidden branch that remains dormant until a certain pattern is found, then overrides the final classifier. Similar structures have now been demonstrated in large-language models [49, 80, 87], confirming a cross-modal threat. Backdoor attacks now extend beyond image classifiers to diffusion models [12] and LLM-powered recommender systems [54].

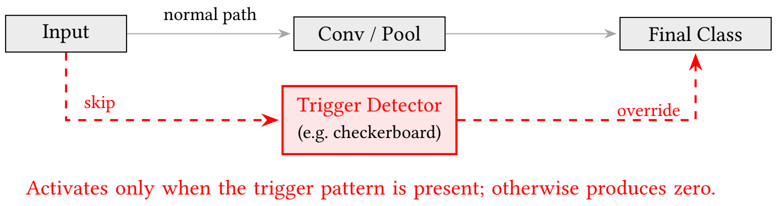

Fig. 2. **Hidden-branch backdoor schematic.** (adapted from Bober-Irizar *et al.* [6]) A light-gray normal path processes inputs conventionally, while a red hidden branch bypasses it. When the trigger pattern is detected, the branch overrides the main features, forcing the attacker-chosen output.

2.2 Motivation for Architectural Threats

Architectural backdoors have gained attention for several reasons:

- **Complex Model Design.** As models grow larger (e.g., vision transformers, large language models (LLMs)), subtle architectural manipulations become harder to detect, providing cover for hidden subgraphs [55, 56].
- **Supply-Chain Exposure.** A compromised compiler, converter, or accelerator can inject logic *after* training, as foreshadowed by Ken Thompson's classic "Reflections on Trusting Trust" compiler Trojan [70] and demonstrated in modern ML contexts by *Shadow Logic* and *ImpNet* [13, 29]. *ImpNet*, in particular, exemplifies a compiler-level backdoor that is both imperceptible and black-box undetectable: by inserting steganographic trigger detectors at the graph Intermediate Representation (IR) level, the attacker creates a dormant conditional pathway that activates only on extremely specific binary input patterns. The model's accuracy on clean inputs remains unchanged, and the trigger cannot be synthesized or detected without reverse-engineering the compiled machine code [13]. Related hardware-Trojan risks are surveyed in [69].
- **Stealth and Persistence.** Because these backdoors reside in the model topology, many classical defenses (e.g. weight pruning or trigger inversion) cannot remove them.
- **AutoML Bias.** NAS often yields shallower, wider architectures with extensive skip-connections [57]. Skip connections themselves are a perfectly benign and widely used design feature (e.g. *ResNets*); they become a security concern only when an attacker leverages that extra routing flexibility to embed hidden logic or alternative pathways. Empirical analyses by Pang *et al.*, [56, 57] show that such automatically searched models are often more vulnerable in empirical benchmarks to adversarial, poisoning, and Trojan attacks than carefully engineered CNNs. This "search-bias" gives attackers a foothold that is largely absent in hand-crafted architectures.
- **Real-World Incidents.** Recent supply-chain breaches, including a dependency hijack in PyTorch nightly builds (Dec 2022), malware-laced models on Hugging Face (Feb 2024), and structural backdoors uncovered in the NIST/IARPA *TrojAI* corpus (Round 15, 2024), a round that, for the first time, pivots the benchmark from input-patch triggers to graph-level structural Trojans, underscoring the field's growing concern[30], show how easily malicious logic reaches production pipelines. In its first six months, Protect AI's Guardian scanner examined ~4.47 million model versions across 1.41 million repositories and flagged about 352,000 suspicious issues affecting 51,700 models [2].

Converging trends such as complex graphs, AutoML shortcuts, and porous supply chains set the stage for the structural backdoor attacks surveyed in the following sections. Understanding them is essential to designing benchmarks and defenses that remain robust as models and toolchains evolve.

Researchers have also highlighted potential hardware Trojans in machine-learning (ML) accelerators [65], which remain invisible to software-level analysis. Deep-learning inference increasingly runs on dedicated chips such as Field-Programmable Gate Arrays (FPGAs), reconfigurable logic fabrics that can be retargeted after manufacture, and Application-Specific Integrated Circuits (ASICs), custom-fabricated devices optimized for a single workload (e.g., Google's TPU dies). Because these accelerators bypass a general-purpose CPU's security stack, even a single malicious logic cell in the bitstream can silently re-enable an otherwise neutralized backdoor. Transformers and other sequence models have likewise been shown to be susceptible to hidden gating sub-layers that activate on specific token patterns [49, 87], broadening the scope of structural threats.

2.3 Key Terminology and Scope

In this survey, we use:

- **Trigger:** A specific input pattern or computational condition that induces malicious outputs. Triggers are designed to be inconspicuous.
- **Malicious Subgraph.** Structural additions or modifications in the computational graph that remain dormant under normal conditions but activate upon a trigger, overriding normal inference [6]. Although many such subgraphs are designed to have no measurable effect until the trigger fires, sophisticated variants can still leak a small bias into the output distribution even in the nominal (untriggered) state, which complicates detection. A concrete early example is *TrojanNet*, which hides an entire classifier inside a host network and triggers it with a tiny pixel key [26].
- **Intermediate Representation (IR)**: A hardware- or framework-specific, low-level graph of operations that sits between a high-level model definition (e.g., PyTorch or TensorFlow code) and the final machine-code or micro-kernel instructions, allowing compilers to optimize and schedule the network for a particular accelerator. ONNX (Open Neural Network Exchange) is an open, hardware-neutral IR standard that captures a model's computational graph, including operators, shapes and data types, so that models trained in one framework can be ported, optimized and executed across many runtimes and devices without re-implementation.
- **Compiler-Level Backdoor:** Logic inserted at the compilation or model-export stage (e.g., into ONNX or other IR), hidden from source-level review [13].
- **NAS/AutoML Backdoors:** Malicious architectures generated by tampering with automated architecture search processes, persisting even when trained on clean data [55, 56].

2.4 Major Architectural Attack Mechanisms

Architectural backdoors can be implanted at various stages, such as model design, compilation, or AutoML-based generation. §3 systematically categorizes these mechanisms, including their threat models and detection challenges.

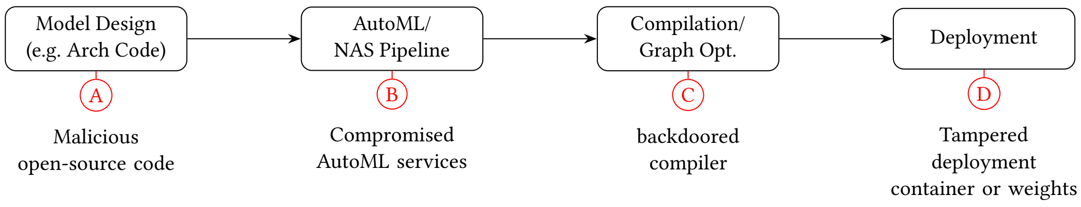
Model Design (e.g. Arch Code)
AutoML/ NAS Pipeline
Compilation/ Graph Opt.
Deployment
A
B
C
D
Malicious open-source code
Compromised AutoML services
backdoored compiler
Tampered deployment container or weights

Fig. 3. **Common attack points in the ML supply chain.**

2.5 Preliminary Defenses and Outstanding Hurdles

Basic static graph inspection or dynamic testing may detect obvious backdoor branches [6, 73], yet more sophisticated designs (e.g., multi-branch or compiler-level) often evade these checks. Weight pruning and trigger inversion, which work against classical backdoors, rarely address structural logic. Formal verification shows promise but faces scalability and complexity barriers, particularly when verifying compiled architectures that diverge from the source definition [70].

2.6 Real-World Motivations and Benchmark Gaps

Supply-Chain Concerns. Malicious compilers, third-party model repositories, and hardware-level backdoors each represent potential insertion points for structural attacks [69]. The *Shadow Logic* proof-of-concept [29] is a vivid example of subgraph-level backdoors slipping past standard checks.

Trojanized Models in the Wild. Since late 2024, Protect AI's public *Guardian* scanner has been flagging architectural backdoor patterns (signatures `PAIT-ONNX-200` and `PAIT-TF-200`) in open-source ONNX and TensorFlow models hosted on the Hugging Face Hub. A six-month progress report states that Guardian has scanned 4.47 million model versions and raised 352,000 unsafe or suspicious findings across 51,700 models, noting that the new detectors "identified additional architectural backdoors" in both formats [2]. One flagged example is the publicly available `SoccerTwos.onnx` file, which Guardian labels "Suspicious-ONNX model contains architectural backdoor." These scanner results show that backdoor-like sub-graphs already appear in real model repositories, although no public analysis has yet demonstrated that the dormant branches can be triggered in practice. We therefore treat them as suspicious architectural backdoors rather than confirmed exploits, while still highlighting their significance for supply-chain security.

LLM-Generated Hardware Trojans. Faruque *et al.* [19] show that LLMs can assist in generating stealthy hardware Trojans. While not compromising the LLM itself, these findings underscore that large models can be used to design advanced backdoors.

Benchmarking Limitations. Datasets like *BackdoorBench* [78] focus mostly on data- or weight-centric backdoors, with limited coverage of structural infiltration and compiler-level vulnerabilities [58]. This gap hinders validation of new defense strategies on realistically compromised architectures, motivating specialized benchmarks for structural threats.

3 Taxonomy of Architectural Backdoors

Following the taxonomy proposed by Langford *et al.* [37], whose user study showed that only 37% of professional reviewers noticed an injected sub-graph, we categorize architectural backdoor attacks based on how and when the adversary injects malicious functionality. Broadly, we identify four categories: (1) backdoors embedded directly into the model's architecture design (via specialized trigger structures within the network, as detailed in §3.1.1); (2) backdoors introduced through a compromised build toolchain or compiler (§3.1.2); (3) backdoors arising from AutoML processes such as NAS (§3.1.3); and (4) hybrid approaches that combine multiple mechanisms to implant backdoors (§3.1.4). We discuss each in turn in the following subsections.

3.1 Comprehensive Attack Vectors and Detection Barriers

Table 2 summarizes four principal architectural backdoor vectors, listing representative mechanisms, threat models and the challenges each poses for detection. Attackers may also blend these vectors; for example, a NAS-generated design can be passed through a tainted compiler that grafts an extra stealth branch post-training. Recent forensic scans confirm this hybrid route: benign architectures

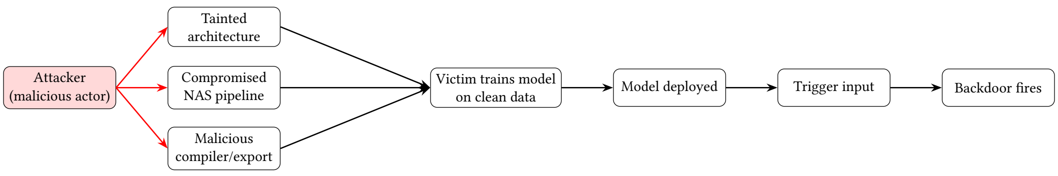

Fig. 4. **End-to-end threat model for architectural backdoor insertion.** A malicious actor can embed a backdoor at three supply-chain stages: tainted architecture code, a compromised NAS pipeline, or a malicious compiler/export tool. Each attack path flows into a seemingly clean training phase; after deployment, the hidden logic is invoked by a crafted trigger, causing targeted misbehavior.

exported via tampered ONNX or TensorFlow pipelines acquired dormant sub-graphs only at serialization time [64].

3.1.1 Detailed Trigger & Subgraph Structures. Architectural backdoors span multiple structural dimensions: some hard-code constant patterns, others exploit operator quirks; some rely on isolated branches while others weave logic into shared paths, and their misbehavior may be targeted or untargeted. Although the literature largely centers on architectural backdoors that force targeted misclassification, the prospect of untargeted, disruptive behaviors should not be ignored. To make the discussion concrete, we categorize the ways a trigger can be embedded in a model's computational graph into three representative patterns:

- **Single-Layer Trigger** : One neuron or micro-layer is modified (or newly inserted) so that it remains dormant on benign inputs yet, upon a specific activation pattern, drives the network toward the attacker's goal. Because the effect is concentrated in one place, this is usually the easiest variant to detect and prune.
- **Subgraph Trigger (A2)** : A single inserted branch or sub-network is grafted onto the host graph. The branch may share early activations but, once its local trigger fires, it bypasses or overrides the main forward path. Bober-Irizar *et al.* introduced this pattern with a checkerboard branch that survives full retraining [6]. Because the subgraph often re-uses benign neurons, isolating and excising it is substantially harder than with a single-layer trigger.
- **Distributed / Interleaved Trigger (A3)** : Trigger logic is split across multiple layers, heads, or even modalities; no individual component appears malicious in isolation. Only a specific constellation of partial triggers activates the hidden route. Classic examples include *TrojanNet*, whose keyed weight permutations embed a covert model [26], and the *Set / Get / Steer* batch-leakage gates of Küchler *et al.* [36] where Set overwrites a victim's output; Get leaks it; Steer subtly biases another user's result—all without direct access. Beyond vision and NLP, Chen *et al.* demonstrate a relation-aware trigger for heterogeneous graph neural networks (HGBA) that inserts only a handful of edges yet achieves near-perfect attack success [9]. Because every fragment piggy-backs on normal activations, graph-diff and activation-clustering defenses frequently fail [8].

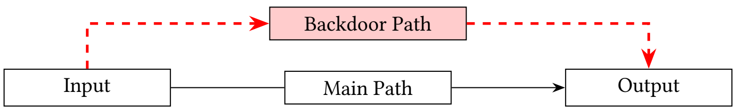

Fig. 5. **Separate-path backdoor.** An isolated malicious circuit (red dashed path) bypasses normal computation entirely and forces the output once its trigger fires.

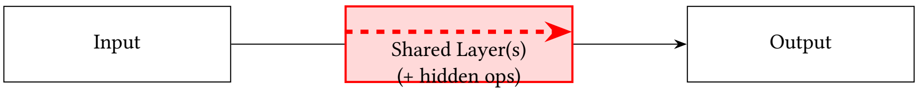

Fig. 6. **Shared-path backdoor.** Malicious computations reside inside the shared layer; the bold red dashed arrow across the top shows the hidden signal, while legitimate processing and label text sit safely below it.

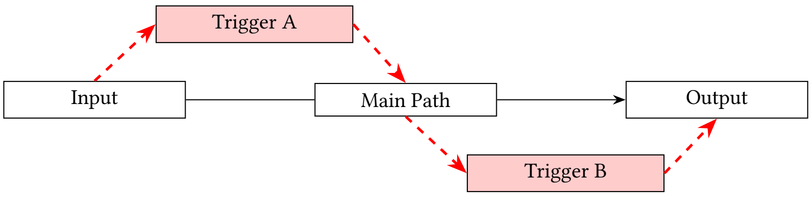

Fig. 7. **Interleaved-path backdoor.** A hidden signal (red dashed line) repeatedly diverges from and rejoins the main computation path, with each hop gated by a different partial trigger. All fragments must activate to corrupt the output.

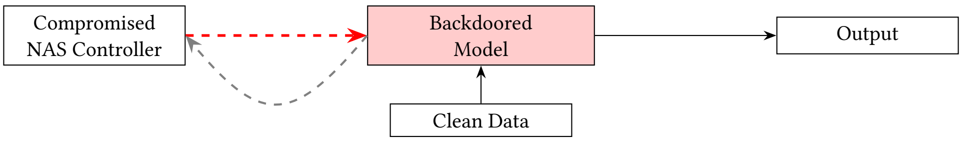

Fig. 8. **NAS-based backdoor.** A compromised AutoML (NAS) controller emits a backdoored model (red), even when trained on clean data. At inference time, the model behaves normally, unless the hidden trigger is present.

3.1.2 Compiler-Based Backdoors. Modern deep-learning pipelines rely on compilers and code-generation tools (e.g., ONNX converters or mobile-deployment frameworks) to translate high-level model definitions into optimized executables. An attacker who compromises this toolchain can inject hidden components during compilation, exactly what *ImpNet* demonstrates by inserting a weight-independent branch that remains invisible to source-level audits [13]. The technique is the modern analogue of Thompson's classic compiler trojan [70]: the compiler surreptitiously adds extra neurons, connections, or conditions that implement a dormant trigger without the developer's knowledge. More recnt work showed 269 real-world ONNX models had data leakage due to poor operator design [36]. The leakage was traced to misuse of the `DynamicQuantizeLinear` operator, which improperly shared state across batch entries Recent findings by Protect AI provide the first confirmed examples of such compiler-level insertions in the wild [64]. Their `PAIT-ONNX-200` and `PAIT-TF-200` disclosures document backdoored models uploaded to public hubs where hidden branches were introduced during ONNX export or TensorFlow SavedModel serialization. These subgraphs, invisible at the source level, lay dormant until specific input triggers were received, demonstrating the feasibility of real-world IR-level manipulation and silent structural sabotage. In a related threat model, Zhu *et al.* [89] show that TensorFlow's export APIs can be abused to embed executable logic, such as shell commands, directly into SavedModel graphs. These logic bombs activate on deserialization, turning AI models into malware containers without affecting model predictions, further expanding the architectural attack surface at serialization time. Qi *et al.* extend this idea to the deployment stage: an adversary can overwrite only a handful of filters to splice a

Table 2. **Expanded 12-Category Taxonomy (Adapted from [37]).** We break each of the four overarching categories, (A) Sub-network attacks, (B) Compiler-based backdoors, (C) AutoML/NAS-based backdoors, and (D) Hybrid attacks, into three subcategories, yielding 12 distinct attack vectors. Each row summarizes the mechanism, threat model, and detection challenges, referencing relevant prior work.

Subcategory	**Mechanism**	**Threat Model**	**Detection Challenges**
(A1) Single-Layer Trigger	A single neuron or layer remains dormant except on a specific pattern, causing misclassification.	Malicious architect or insider with direct control over model design.	The local trigger can be subtle yet persistent, evading naive weight inspection [6].
(A2) Subgraph Trigger	A small subnetwork is embedded for the backdoor, sometimes partially used in normal computation.	Attacker modifies the official architecture code or design blueprint.	Because a backdoor sub-network can reuse neurons that also serve normal computation, identifying its malicious role is substantially harder than for a single rogue layer [6].
(A3) Distributed / Interleaved Trigger	Trigger logic is spread across multiple layers or nodes, each benign in isolation. Examples range from TrojanNet's hidden-weight permutation model [26] to the recent batch-context leakage gate that copies features across examples within a single inference batch [36].	Attacker designs complex network topologies with multiple partial triggers, structural switches, or cross-example paths.	Requires holistic graph analysis to detect; no individual element exhibits an obvious malicious pattern [26].
(B1) Malicious IR-Rewriting	*ImpNet* inserts a hidden branch directly into the IR during compilation.	Attacker controls the build/export pipeline (cf. Thompson's "trusting-trust").	Source code looks clean; the injected branch is visible only in the compiled graph [13, 70].
(B2) Compiler-Level Operator Injections	Standard ops (Conv, ReLU, etc.) replaced with modified versions that hide triggers.	Root-level access to compiler or operator libraries.	Subtle operator changes do not alter normal accuracy; triggers remain dormant until specific input [29].
(B3) Trojaned Export/ Serialization	A malicious export tool adds extra nodes or branches when converting to ONNX/TFLite.	Attacker can tamper with final serialization routines post-training.	The "deployed" model differs from the validated training model, rarely re-checked in practice [29, 64].
(C1) Malicious Reward Shaping in NAS	NAS objective or reward is altered so that backdoor-friendly designs get higher scores.	Attacker modifies the AutoML pipeline to favor certain sub-blocks or triggers.	Models appear optimal on clean data, yet contain dormant subgraphs by design [55].
(C2) Poisoned Search Space	Malicious building blocks or candidate layers in the search space embed dormant triggers.	Third-party or coerced library contributor seeds hidden modules.	The hidden module is simply "selected" by the NAS algorithm; easy to miss unless the search space is audited [56].
(C3) Injected Sub-Block Shortcuts	Small sub-blocks with conditional shortcuts are inserted in the final architecture.	Compromised AutoML code that automatically includes the sub-block in each candidate design.	Disguised as normal architectural blocks, they only activate under a special input pattern [55].
(D1) Architecture + Compiler Synergy	Adversary plants a structural backdoor, then obfuscates it further with malicious compile steps.	Full pipeline control (model design + compiler).	Redundant triggers: removing one layer of the backdoor might leave another intact [13].
(D2) Architecture + Data Poisoning	Mild data poisoning supports a hardware-level or structural backdoor.	Attacker can tamper with both training data and the architecture.	Defenses focusing only on data anomalies or only on structural audits can miss the combined effect [56].
(D3) Multi-Stage Obfuscation	Multi-phase approach: malicious design + data poisoning + compile-level changes.	Well-resourced attacker infiltrating every stage of the model lifecycle.	Each malicious layer is benign on its own; the final deployed model exhibits stealthy, robust triggers [6].

one-channel hidden subgraph into a trained CNN, achieving $>$ 99% attack success rates (ASR) on *ImageNet* while reducing clean accuracy by less than 2%, all without training data or gradients [62].

3.1.3 AutoML/NAS-Based Backdoors. Beyond manual design and compiler manipulation, recent work has revealed that AutoML processes themselves can introduce backdoors. AutoML pipelines, especially NAS, can be deliberately manipulated to yield models with hidden architectural backdoors. Pang *et al.* [55, 56] explicitly introduce the exploitable and vulnerable arch search *(EVAS)* and show that, by adding a malicious reward term, an adversary can steer NAS to select architectures containing a dormant "shortcut" subgraph which activates on a specific trigger pattern. This AutoML-based backdoor attack is especially insidious because it does not require any explicit tampering with training data or model parameters. What survives is the capacity for malicious behavior: the hard-wired branch remains present, and standard training loss will quickly re-learn suitable weights, so the attacker needs little or no additional effort to reactivate the payload. Moreover, since the compromised model's weights and training data may appear entirely benign, such architecture-level backdoors naturally evade many conventional defenses that focus on detecting anomalies in the training process or parameter values. This emerging class of attacks highlights a new security risk in AutoML pipelines, calling for heightened scrutiny of NAS-generated models. Although the Protect AI incidents involved post-training insertion, they underscore that even cleanly designed NAS-generated architectures can be compromised during export, highlighting the need for end-to-end validation of AutoML pipelines.

3.1.4 Hybrid and Combined Modes. Hybrid or multi-stage backdoor attacks could pose a particularly potent threat because they can evade defenses designed to detect single-stage insertions. For instance, an attacker might first leverage NAS to introduce subtle vulnerabilities during architecture search and subsequently manipulate the compiler or model-export phase to further conceal or strengthen the malicious logic [13, 56]. Such multi-stage approaches reflect real-world attack sophistication, requiring defenders to adopt equally comprehensive detection and mitigation strategies. Attackers can combine methods, for instance feeding a NAS-generated graph through a tainted compiler that grafts a stealth branch while also performing mild data poisoning during training. Such hybrid backdoors blend architectural and training-time vectors, allowing one component to survive even if the other is mitigated. Because these multi-stage exploits bypass defences that target a single insertion point, effective protection must monitor the entire model life-cycle.

3.2 Expanded 12-Category Taxonomy (Adapted from [37])

Building on the twelve-subcategory framework of Langford *et al.* [37], we further break each family into three sub-types, yielding twelve distinct architectural attack vectors. Table 2 summarizes these subcategories along with their mechanisms, threat models, and detection challenges [6, 13, 16, 55, 56, 70].

4 Detection of Architectural Backdoors

Architectural backdoors often resist established detection methods, which typically focus on suspect weight distributions or poisoned training samples, because topology-level backdoors evade traditional anomaly checks. For instance, activation clustering [8] or spectral signatures [71] detect unusual neuron activations triggered by poisoned data, but an architectural backdoor can remain dormant on clean data, producing no detectable activation anomalies. Similarly, methods like *Neural Cleanse* [73], designed to reverse-engineer visible single-input triggers, may fail to detect architectural triggers that depend on internal multi-branch conditions or composite input patterns. Recent research into "defense-aware" attacks specifically aims to evade such inversion methods [49]. Thus, specialized detection strategies for architectural threats are critically needed. In this section, we provide:

- Key Definitions and Motivations: Explaining why topology-level backdoors evade traditional anomaly checks.

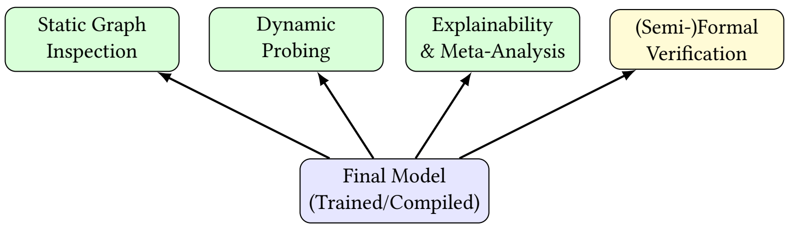
Static Graph Inspection
Dynamic Probing
Explainability & Meta-Analysis
(Semi-)Formal Verification
Final Model (Trained/Compiled)

Fig. 9. **Overall detection pipeline.** Defenders can apply one or more methods (static, dynamic, explainability, and (Semi-) formal verification) to reveal suspicious subgraphs.

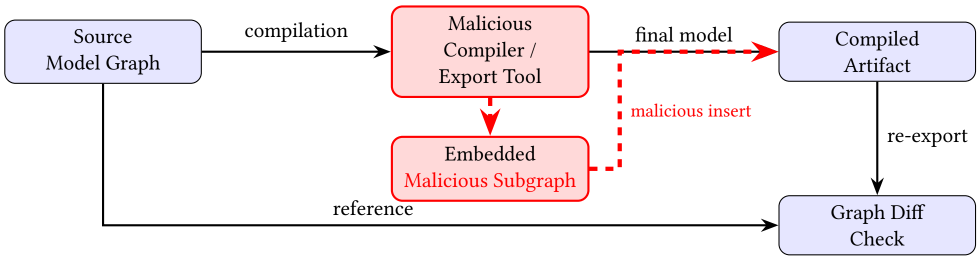

Fig. 10. **Compiler-level insertion and static inspection.** A compromised compiler injects a hidden subgraph into the compiled artifact. A subsequent *graph-diff* step detects the discrepancy by comparing the reference model graph with the potentially compromised artifact.

- Structured Detection Approaches: Grouped into static graph inspection, dynamic/trigger inversion, explainability/meta-analysis, and formal verification methods.
- Challenges, Benchmarks, and Metrics: Highlighting the real-world obstacles to detection and the limited coverage of existing benchmarks.

Architectural backdoors are tackled via four complementary families: (i) *static graph inspection* (§4.1), (ii) *dynamic trigger discovery* (§4.2), (iii) *explainability & meta-analysis* (§4.3), and (iv) *(semi-)formal verification* (§4.4). Each family trades scalability against completeness, so a layered pipeline (Fig. 9) is recommended in practice.

4.1 Static Graph Analysis and Model Introspection

Static graph analysis inspects the exported model (ONNX, TorchScript, etc.) for sub-paths or operators that diverge from the intended architecture. Architectural backdoors typically surface as extra gating layers, direct input-to-output bypasses, or custom ops absent from the high-level source [6, 55]. Figure 10 illustrates how a clean source can compile into a binary with an implanted subgraph. Practically, integrating static analyzers that systematically compare exported model formats (e.g., ONNX or TensorFlow GraphDef) against the original intended architecture can significantly streamline anomaly detection. Static inspection is lightweight and flags obvious structural backdoors, but it often misses obfuscated or deeply-interleaved logic. For example, automated checks for additional layers, unexpected gating logic, or custom operations not defined in the original architecture can be implemented. Maintaining accurate original architecture specifications as reference artifacts during model development is recommended as a best practice for facilitating such comparisons.

Limitations of pruning and purification-based defenses. Classical defenses such as fine-pruning [46] aim to remove dormant neurons by eliminating those with low activation on clean data, assuming backdoor logic is encoded in rare weight activations. While effective against weight-based backdoors, this strategy is poorly suited to architectural backdoors, where the malicious logic may reside in the topology itself, e.g., in an extra branch or gated path that is never activated on clean inputs. Recent work further reveals the brittleness of purification-based methods: Gradient Tuning Backdoor Attack++ (*GTBA++*) [51] demonstrates that models, even after pruning or adversarial unlearning, can rapidly reacquire high attack success rates when exposed to limited poisoned data. Similarly, the ASR-Proof evaluation framework [50] demonstrates that many existing defenses only superficially reduce ASR, failing to eliminate the latent backdoor mechanism. Consequently, attackers can quickly reactivate the backdoor using carefully constructed adaptive queries. These results underscore the need for architecture-aware defences that explicitly reason about structural logic and supply-chain integrity. Static analysis is, therefore, an expedient first line of defense. However, high-value deployments should complement it with the dynamic probing techniques of §4.2, which can activate stealthy paths that static diffs overlook.

4.2 Dynamic Probing and Trigger Inversion

Dynamic techniques actively search for inputs that route execution through a hidden subgraph. Unlike static diffing (§4.1), they treat the model as a black box and watch for abrupt confidence shifts or label flips. Dynamic probing complements static graph inspection by attempting to uncover stealthy backdoor logic through targeted perturbation of inputs or reverse-engineering potential triggers from model responses. While static approaches analyze the model structure directly, dynamic methods provide evidence through empirical behavior.

Fuzzing. This randomized approach perturbs model inputs and monitors outputs for unexpected spikes in softmax entropy or abrupt shifts in predicted class probabilities [63]. Recent fuzzing-based methods, such as Runtime Oracle-guided Search for Backdoor Analysis *(ROSA)* [34], use guided fuzzing to discover backdoor triggers in traditional software. Although primarily developed for software vulnerability discovery, analogous fuzzing techniques are gaining attention for ML backdoor detection. Nonetheless, exhaustive fuzzing quickly becomes computationally prohibitive, particularly for complex, composite triggers, highlighting the need for advancements towards multi-modal fuzzing strategies.

Trigger Inversion. Neural Cleanse [73] and its extensions [84] frame backdoor discovery as an optimization problem, aiming to identify the smallest perturbation necessary to consistently produce targeted misclassification. These methods initially proved effective for single-input triggers, especially in image classifiers, but encounter significant difficulties with multi-branch or composite triggers. Recent research into "defense-aware" backdoors, such as those demonstrated by Miah *et al.* [49], has specifically developed strategies to evade trigger inversion, underscoring the limitations of relying solely on inversion techniques.

Data-Limited Trigger Search. Recent developments have addressed detection scenarios under harsh practical constraints, notably *DeBackdoor* by Popovic *et al.* [61]. Operating with only black-box access and extremely limited clean samples (less than 1% of the training dataset), *DeBackdoor* employs a deductive search across an extensive trigger hypothesis space (e.g., shapes, patches, blended patterns). By iteratively optimizing a smoothed attack-success objective through simple forward passes, *DeBackdoor* reconstructs potential triggers without needing large validation sets or direct weight access. Experiments involving multiple architectures, datasets, and attack types demonstrate near-perfect detection accuracy, highlighting its viability for auditing third-party models and uncovering input-dependent manifestations of architectural backdoors. Architectural triggers that alter only logit magnitudes, or otherwise induce minimal output drift, may still dodge current fuzzing and entropy-based heuristics, underscoring the need for more sensitive or adaptive oracles.

4.3 Explainability–Based Detection and Meta-Analysis

Explainability. SentiNet [11] applies Gradient-weighted Class Activation Mapping (Grad-CAM) to locate the most influential regions of an image for a given model decision. Backdoored models often reveal a distinctive heatmap pattern when the trigger is present, as the malicious subgraph sharply concentrates the relevance on the trigger area. Saliency masks or attention weights can thus expose anomalous focus, especially if multiple inputs share a suspiciously consistent activation region. Learning-based meta-detectors such as *MNTD* (Kolouri *et al.*, S&P 2021) train a binary classifier on model query–response behavior and achieve high AUC in distinguishing Trojaned from clean models without requiring trigger access [35].

Meta-analysis. Meta Neural Trojan Detection (MNTD) [35] trains a black-box meta-classifier on a labelled dataset of clean and backdoored models. The classifier then evaluates an unseen model by observing its query–response behavior. A related method is *ABS (Artificial Brain Stimulation)* [47], which perturbs internal neurons and evaluates outputs to infer latent backdoor presence. While effective on conventional image classifiers, these approaches depend heavily on training coverage and generalization assumptions. Recently, Shen *et al.* introduced *BAIT* (Backdoor scanning by Inverting the Target) [66], a meta-level detection method targeting large language models. *BAIT* uses a novel inversion pipeline that traces attacker-desired completions back to hidden trigger patterns embedded in the model's graph or behavior. Unlike earlier methods that require explicit trigger injections, BAIT reverses from suspicious outputs to uncover latent structural logic, enabling scalable detection of backdoor pathways even in fine-tuned or instruction-tuned LLMs. As LLMs increasingly incorporate adapters and LoRA modules, parameter-efficient fine-tuning also opens new vectors for architectural backdoors. *PEFTGuard* [68] bridges meta-analysis and explainability by inspecting parameter-efficient fine-tuning (PEFT) modules for hidden decision logic and providing visual explanations of their graph-level impact. Although designed for parameter-efficient settings, its structured introspection pipeline is directly relevant to architectural risk modeling in LLM-class systems. Explainability excels when a trigger leaves a spatial or statistical footprint, whereas meta-analysis trades false positives for scalability to unseen models. Newer tools like *BAIT* and *PEFTGuard* signal a promising shift toward inversion-based explainability that generalizes across architectures and formats.

4.4 Formal Verification and Semi-Formal Methods

Formal verification frameworks, such as *Reluplex* [32] and its successor *Marabou* [33] express the backdoor question as: "Is there any input that is almost identical to a normal one yet still flips on the hidden branch?" Here "almost identical" means the input sits inside a very small neighborhood, often called an *ℓ_p ball*, around a clean reference example. For images, two common interpretations of that neighborhood are (i) **per-pixel bound** (no individual pixel is changed by more than a tiny value) and (ii) **overall-energy bound** (the total Euclidean change across all pixels is tiny). If the solver can show that every such near-clone leaves the backdoor branch dormant, the model is certified safe within that radius. If it cannot, the tool returns a concrete offending input, effectively uncovering the trigger [60]. A deeper scalability analysis appears in § 7.1.

4.5 Strengths & Limits of Current Detectors

The state of the art in detecting architectural backdoors is still hemmed in by four interlocking hurdles. First, multi-trigger complexity frustrates current tools: when malicious logic is distributed across several branches or layers it can demand a specific constellation of signals before activation [37, 45, 84]. Second, compiler obfuscation undermines source-level audits; an apparently clean model definition may compile into an intermediate representation that hides a backdoor subgraph, a strategy dramatized by the *Shadow Logic* proof-of-concept [29]. Third, both dynamic probing and (semi-)formal verification carry a steep computational cost once models approach modern scale, so practitioners face a trade-off between coverage and run-time analysis. Finally, genuine progress is hampered by benchmark scarcity: most public datasets still revolve around pixel- or label-poisoned CNNs, leaving structural exploits to small ad-hoc collections [6, 55, 56]. Because architectural attacks typically leave clean-set accuracy untouched, the community relies on indicators such as true-positive rate (TPR), false-positive rate (FPR), and attack-success-rate reduction (ASR) achieved after mitigation. Benchmarks like *BackdoorBench* [78] and government-sponsored *TrojAI* evaluations [72] report these numbers consistently. However, even top detectors in TrojAI primarily demonstrate effectiveness on data-poisoning-based backdoors; these challenges have not extensively tested hidden architectural or compiler-level insertions, leaving a critical evaluation gap for real-world structural threats. An additional community resource is *TrojanZoo*, an open-source corpus of backdoored and clean models released by Pang *et al.* [58]; it provides graph-level ground truth for CNNs and Transformers and is now widely used to sanity-check detection pipelines. A credible next generation should cover at least three scenarios that today's datasets omit altogether: (i) multi-branch gating models in which individual paths look benign in isolation; (ii) backdoors introduced only at compile time, visible in the binary graph but absent from the high-level source; and (iii) transfer-learning persistence tests that check whether a architectural backdoor survives when the model is fine-tuned on a new domain [74]. Incorporating these cases would close the evaluation gap and give forthcoming detection methods a realistic proving ground.
Table 3 summarizes the complementary strengths and blind spots of static, dynamic and formal approaches. Because no single technique is fool-proof, readers should see the consolidated research gaps in §7. The next section (§5) turns to repair: how to neutralize flagged sub-graphs, restore benign accuracy, and harden the pipeline against future inserts.

Table 3. Comparison of static, dynamic and formal detection methods.

Approach	**Strengths**	**Limitations**
Static graph	Lightweight; reveals overt gating or custom ops without training data	Blind to subtle, distributed, or compiler-time inserts
Dynamic probing	Confirms malicious behavior on crafted inputs	Random/gradient search struggles with multi-condition triggers or heavy obfuscation (e.g., SGBA decoys [28])
Formal verification	Provable absence of backdoors within bounded input domain; yields counter-example if proof fails	Currently unscalable to large, multi-branch models; misses spectral/composite triggers; invalidated by source/IR drift
Hybrid pipeline	Layers static, dynamic and partial proofs for broader coverage	Requires orchestration across tools; no standard benchmark for supply-chain scale yet

5 Mitigation and Model Repair

Architectural backdoors (§4) resist naive mitigation because the malicious logic is structurally "wired" into the model's topology. Unlike weight backdoors that one can 'forget' by fine-pruning or retraining, a structural backdoor must be surgically removed from the network architecture. This process is complicated by multi-branch logic that scatters gates across layers [55], topological embedding that survives weight resets, rare trigger activations that evade fine-tuning gradients, and the persistent risk of re-insertion via compromised supply chains. This section surveys practical, graph-level, and supply-chain measures tailored to excise or disable these sophisticated structural threats, with particular emphasis on multi-branch and compiler-time attacks.

5.1 Subgraph Pruning and Removal

Static diffing (§4.1) or dynamic probing (§4.2) first localize a suspect path, potentially using guided fuzzers such as ROSA [34], then excise that subgraph and optionally fine-tune to recover accuracy. Classical fine-pruning of neurons[46] or dataset filtering alone leave the graph intact and thus fail against topological attacks. Bober-Irizar *et al.* report that excising the checkerboard branch slashed ASR from 100 % to 2 % (98 %) while leaving clean accuracy intact [6] We suggest practitioners first use static diff (§4.1) or dynamic probing (§4.2) to identify suspicious subgraphs, excise them, and immediately re-check model accuracy. If attack success remains partially elevated (e.g., around 40%), it indicates additional hidden branches still exist. Practitioners must then iteratively remove these residual branches until attack success is fully neutralized. Complementing these approaches, recent work by Bajcsy and Bros [4] introduces a web-based simulation playground that lets practitioners plant, trigger, and defend against cryptographic, including checksum based, architectural backdoors. The sandbox supports realistic plant-and-defend cycles, enabling researchers to validate and stress-test backdoor detection and proximity-analysis defenses.

Table 4. **Subgraph removal at a glance**

	Strengths	**Caveats**
Excise single branch	Directly disables known route; no full re-train needed	Accuracy loss if branch intertwines with benign features
Multi-branch pruning	Cuts distributed triggers when combined with iterative search	Residual gates survive if any branch escapes detection
Compiler-aware pruning	Checks IR after build to confirm removal	Fails if toolchain is still compromised; must repeat per build

A hybrid mitigation pipeline should therefore: (1) detect all suspect paths; (2) prune or redirect them; and (3) re-export the model under a trusted, signed compiler to prevent silent reinsertion. Subsequent sections cover adversarial unlearning and supply-chain hardening that complement graph excision.

5.2 Adversarial Unlearning and Re-Training

Once a trigger or suspect gate is located (§4), adversarial unlearning seeks to purge its influence by re-training the model on that trigger with the correct label. Early work such as *Neural Cleanse* searches for a minimal input perturbation that flips the label and then fine-tunes on the resulting corrected pair [73]. Methods like Implicit Backdoor Adversarial Unlearning (*I-BAU*) [83], Anti-Backdoor Learning (*ABL*) [42], and Neural attention distillation *(NAD)* [43] extend this approach to broader trigger types, while more recently, neural-collapse cleansing restores a backdoored network by realigning its feature geometry with that of a clean reference model [25]. While *NAD* assumes access to a clean teacher, *PEFTGuard* [68] facilitates backdoor localization for parameter-efficient settings. It automatically inspects LoRA and adapter modules for potential backdoor logic and provides saliency-style visualizations of their effect on model behavior. *PEFTGuard* combines lightweight structural analysis with feature attribution to identify which modules disproportionately influence attacker-desired outputs. Although designed for PEFT scenarios, its logic-based analysis generalizes well to architectural backdoor repair tasks involving sparse or modular subgraphs. Adversarial unlearning is effective primarily for backdoors that actively influence model parameters or decision boundaries. Architectural backdoors, especially those rarely activated, present limited gradient signals during fine-tuning, rendering unlearning incomplete. Composite or multi-condition triggers require exposing every trigger condition, further complicating remediation efforts. More fundamentally, cryptographic backdoors compiled into the model architecture, as shown by Draguns *et al.* [15], may resist all current unlearning methods. Their encrypted trigger-payload circuits cannot be reliably activated even through latent adversarial training (LAT), which is widely viewed as the state of the art in backdoor elicitation. This suggests that adversarial fine-tuning cannot mitigate backdoors that are non-continuously differentiable or intentionally obfuscated at the computational-graph level. These limitations motivate complementary, lightweight defenses such as pruning, attention distillation (§5.3), and runtime monitoring (§5.4). Recent work on securing transfer learning pipelines introduces proactive filtering to exclude compromised components before they are learned. *T-Core Bootstrapping* [85] identifies trustworthy neurons and data instances early in fine-tuning, mitigating both architectural and parameter-based backdoors in pre-trained encoders. By constraining the transfer learning process to "core-safe" features, *T-Core* reduces the risk that a dormant backdoor is inherited from a contaminated upstream model or dataset. This strategy is particularly valuable when retraining on sensitive downstream tasks where subtle misbehavior could evade manual inspection.

5.3 Attention Distillation and Model Surgery

NAD aligns a potentially backdoored student model to a clean teacher by matching intermediate attention maps [43]. Li *et al.* demonstrated that *NAD* significantly recovers clean accuracy. However, *NAD*'s effectiveness depends critically on having access to a trusted clean teacher model of similar architecture, a prerequisite often unavailable in practice. Further, architectural backdoors that span multiple distributed branches may evade partial realignment or isolated surgical interventions, necessitating more extensive model surgery or combined mitigation strategies. If the malicious logic sits in identifiable heads or layers, simply disabling those modules and fine-tuning can work [37, 55]. Multi-branch attacks often do not localize neatly into one identifiable head or layer, requiring more extensive surgery or combination with other mitigation strategies to avoid significant accuracy loss. Combined with subgraph excision (§5.1), these attention-centric methods form the second line of defense before costly full re-training or supply-chain rebuilds.

Table 5. **Attention-based repair: pros and cons**

	Strengths	**Caveats**
NAD	Systematically realigns internal attention; no full retrain if teacher available	Needs a clean teacher of similar architecture
Layer/Head removal	Fast; surgically targets known gate	Accuracy drop if gate overlaps benign function; ineffective on interwoven logic

5.4 Runtime Monitoring and Canary Testing

Stealthy architectural backdoors may activate only on rare inputs, so *in-production* monitoring must complement offline defences.

Entropy / anomaly monitors & canaries. Integrate STRIP-style entropy checks [21] and inject crafted canaries that mimic likely triggers [13]. Each flags a live sub-graph, yet composite or distributed triggers [37, 79] can evade both, so runtime monitoring should be viewed as a layer, not a fix.

Input randomisation. Pixel or gate noise can break exact-match triggers [55, 84]; robust backdoors survive, stressing the need for multi-pronged runtime guards.

Domain-specific runtime shields. GraphProt [81] defends black-box graph classifiers by (i) clustering each input graph, (ii) sampling purified sub-graphs, and (iii) ensembling their predictions, cutting ASR by up to 90% on six datasets with around 1–2% accuracy loss. Because it needs no retraining or weight access, *GraphProt* illustrates practical, domain-aware runtime shielding.

LLM-assisted filtering for recommender systems. P-Scanner [54] counters BadRec, a 1%-poison backdoor that forces any tokenized item to be recommended, by fine-tuning a large language model to spot semantic anomalies. It removes > 90 % of poisoned samples across three real-world datasets, restoring accuracy with negligible overhead.
Runtime monitoring flags, but does not remove, a backdoor; it is most effective alongside the offline mitigations of §§5.1–5.3. Structural threats already permeate diffusion models too [12], underscoring the need for broad, layered runtime defenses.

Table 6. **Runtime checks: strengths and caveats**

	Strengths	**Limitations**
Entropy/anomaly	Low overhead; continuous monitoring	False negatives if outputs stay “plausible”
Canary testing	Direct confirmation of malicious path	Requires guessing the trigger; composite keys may evade
Randomization	Cheap perturbation defense	Advanced backdoors can be perturbation-robust

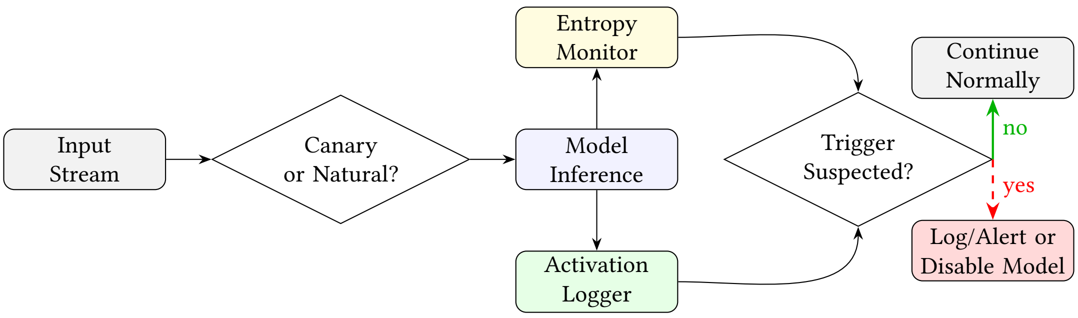

Fig. 11. **Runtime Backdoor Detection Flow (rotated).** Even after offline mitigation, defenders can feed canary inputs or monitor normal traffic for suspicious triggers. If detected, the model can be logged or sandboxed; otherwise, inference continues normally.

5.5 Supply-Chain Assurance and Trusted Compilation

Even after local repair, a tainted build pipeline can surreptitiously re-insert malicious logic at export time. Recent incident reports from Protect AI (`PAIT-ONNX-200` and `PAIT-TF-200`) uncovered ONNX and TensorFlow SavedModel artefacts whose hidden trigger branches were added only during serialization [64]. Zhu *et al.* further show that TensorFlow's export APIs can embed shell-executing ops into the graph, transforming an ordinary model into a malware container without altering predictions [89]. These findings underscore that robust assurance must span the entire pipeline: *source* $\longrightarrow$ *compiler IR* $\longrightarrow$ *runtime* $\longrightarrow$ *hardware*. Deterministic builds enable IR differencing, and recent static–taint methods such as Batch Isolation Checker attach a short, machine-checkable proof to each lowering pass [36]. See §7 for a detailed checklist on reproducible builds and IR-level provenance.

Malicious AutoML pipelines. Attacks such as *DarkNAS* embed backdoors during architecture search when the reward signal is under adversarial control [55, 56]. Supply-chain checks must therefore verify AutoML artifacts as well as compiler outputs.

TensorFlow logic-bomb exploits. Zhu *et al.* demonstrate that only a handful of TensorFlow API calls, inserted during export, can plant executable payloads inside a SavedModel graph [89]. The malicious ops fire at deserialization time, executing arbitrary commands while leaving model accuracy intact. Because the backdoor lives in the standard SavedModel format, it evades conventional weight checks and illustrates why supply-chain audits must include static operator allow-lists and sandboxed loading for untrusted artefacts.

Hardware Trojan risk. Compromised accelerators can manifest a backdoor at run-time without changing weights [65]. End-to-end assurance echoes SoC security practice [69]. A key takeaway from the Logic backdoor proof-of-concept [29] is that stealth subgraphs added post-training may go unnoticed in source code and only be revealed through IR differencing. This highlights compiler-time insertion as a potent and easily overlooked infiltration vector.

Replicated Execution for Outsourced Training. Replicated execution across multiple non-colluding cloud providers [31] can verify outsourced training jobs by cross-comparing intermediate checkpoints for anomalies. Although redundant execution increases costs, for critical models, training smaller subset models across diverse providers can effectively identify suspicious insertions. Federated learning frameworks, which already employ partial model averaging and integrity checks, provide a relevant precedent illustrating this approach's feasibility in practice.

5.6 Concluding Summary

Architectural backdoors reside in *topology*, so defenders must layer remedies: subgraph excision, adversarial unlearning, attention surgery, runtime checks, and critically-supply-chain signing. Table 7 summarizes how each technique addresses architectural backdoor; Figure 12 gives one possible decision flow.

Open problems. (1) Fully integrated multi-trigger attacks still defeat current tools. (2) Scalable formal repair for billion-parameter graphs remains elusive (§4.4). (3) Pipeline re-introduction demands continuous verification, not one-off cleaning. (4) Benchmark gaps slow progress; few suites model compiler-time insertion or AutoML backdoors. Continued cross-layer research is needed for durable defense: this must span graph analysis, automated repair, secure AutoML, reproducible builds, and hardware attestation. Addressing these open problems will require sustained interdisciplinary collaboration among researchers, industry practitioners, and regulatory bodies, ensuring

Table 7. **Comparison of Mitigation Methods for Architectural Backdoors.** Each approach tackles a different slice of backdoor logic; multi-branch or compiler trojans often need a mix of these methods.

Method	**Key Advantages**	**Main Limitations**
Subgraph Removal	Physically excises identified malicious routes; low data cost; effective if the backdoor path is separate.	Fails for integrated triggers woven into normal blocks; may hurt accuracy if removal is too extensive.
Adversarial Unlearning & Re-Training	Disassociates known triggers from attacker outputs; flexible across backdoor types; I-BAU cuts overhead.	Must activate every trigger to "unlearn" it; multi-trigger logic can persist; costly for large models.
Attention Distillation & Surgery	Re-aligns suspicious layers/heads; partial fine-tune preserves benign performance if a clean teacher exists.	Distributed or multi-branch trojans may evade partial realignment; need a precise mapping of malicious parts.
Runtime Monitoring & Canary Testing	Watches models in deployment; can catch unforeseen triggers; little extra training.	Does not remove the trojan; stealthy gating can mimic normal outputs and slip past anomaly checks.
Supply-Chain Assurance	Blocks re-injection; guarantees final artefact matches a trusted build; covers AutoML or compiler trojans.	Adds organisational overhead; needs signed/reproducible builds; hardware Trojans remain a separate threat.

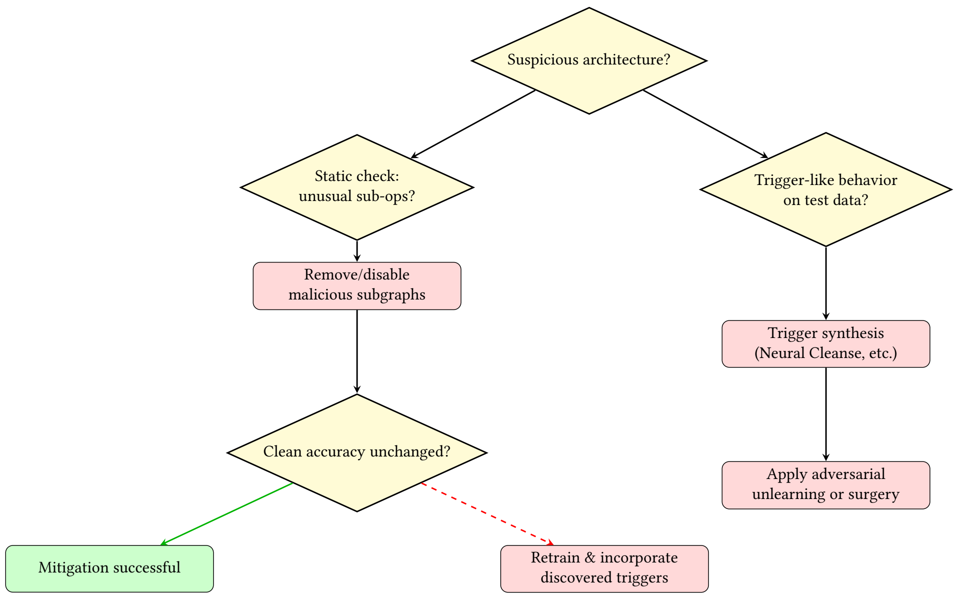
Suspicious architecture?
Static check: unusual sub-ops?
Trigger-like behavior on test data?
Remove/disable malicious subgraphs
Trigger synthesis (Neural Cleanse, etc.)
Clean accuracy unchanged?
Apply adversarial unlearning or surgery
Mitigation successful
Retrain & incorporate discovered triggers

Fig. 12. **Decision flow for mitigating architectural backdoors.** Starting from suspected infiltration, defenders apply static or dynamic checks to pinpoint malicious subgraphs or triggers. They then perform subgraph removal, adversarial unlearning, or attention surgery. If clean accuracy is significantly impacted, more extensive re-training follows. Finally, ensuring a trusted supply chain (§5.5) prevents backdoor re-insertion.

that advancements in graph-level verification, runtime defenses, secure compilation, and hardware integrity checks are tightly integrated into a robust, end-to-end defense ecosystem against architectural backdoors.

6 Benchmarks, Datasets, and Empirical Evaluations

Having surveyed detection (4) and mitigation (5), we now examine how well those defenses hold up in practice. Although many benchmark suites exist for data- or weight-centric backdoors, few systematically address structural or compiler-level attacks. This section reviews current resources, highlights gaps, and suggests directions for multi-branch and supply-chain-aware evaluations.

6.1 Current State of Backdoor Benchmarking

Standard backdoor metrics (detection TPR/FPR, ASR reduction) still apply, but evaluations must treat the trigger as a latent sub-graph or operator sequence rather than a visible pixel patch. *TrojAI* Round 15 was the first public benchmark to distribute ONNX binaries that contain hidden branches requiring structural detection or neutralization, all without relying on poisoned weights [30]. Nevertheless, coverage is still incomplete: pure compiler-time inserts [13], multi-branch triggers, and post-fine-tune reactivation remain largely untested. Protect AI's PAIT scans flag real ONNX/SavedModel artefacts with hidden branches (e.g. `PAIT-ONNX-20`, `PAIT-TF-200`) and publish full metadata [2]. Incorporating a "PAIT track" in future rounds (*TrojAI* R14, *Backdoor-Bench*-X) would inject live, diverse compiler-time trojans and tighten realism. *BackdoorBench* adds only a few gating-op tests; *TrojAI* and the NeurIPS'20 challenge target data/weight triggers [53]. Ad-hoc academic sets rarely include compiler or multi-branch inserts [6, 57], nor do they assess fine-tuning persistence or supply-chain infiltration. Table 8 summarizes present coverage.

Table 8. Architectural coverage of today's backdoor benchmarks

Benchmark	**Main focus**	**Struct./Compiler coverage**
TrojAI (IARPA)	Data-poison, weight triggers	Minimal; no compiler path
BackdoorBench	Data/weight + a few gating ops	No multi-branch or IR sabotage
NeurIPS TD Challenge	Competition on parameter triggers	None (structural absent)
Academic sets	Small crafted examples	Sporadic, ad-hoc structures

Due to the lack of coverage for architectural backdoors in most backdoor benchmarks, many architectural-backdoor studies rely on ad-hoc model sets. Bober-Irizar *et al.* [6] used about 20 AlexNet variants with a checkerboard trigger, while Pang *et al.* [55] manipulated a NAS pipeline for CIFAR. Both illustrate key ideas but lack scale or diversity comparable to data/weight-centric benchmarks. These examples underscore an urgent need for more extensive, standardized evaluations tailored to structural infiltration.

6.2 Measuring Success in Architectural Backdoor Mitigation

Mainstream benchmarks typically report true/false positive rates, ASR reduction, and resource overhead. Although these metrics work for architectural backdoors, they do not capture unique complexities like multi-branch triggers or domain-shift reactivation. A hallmark of architectural backdoors is their survivability, they often persist after full re-training. For instance, backdoor studies based on the *CIFAR-10* dataset demonstrated that weight-based backdoors can be "forgotten"

by retraining, but model architecture backdoors (MABs) [6] and operator-based [37]) retained attack success even after full retraining. Similarly, Gu *et al.* [24] showed that a trigger in a traffic-sign classifier persisted after fine-tuning on a new task, an aspect seldom tested in standard frameworks. Attack success for models with architectural backdoors remains high in these studies because the malicious route is encoded in the model's graph, not just in weights.
Beyond detection, effective mitigation needs benchmarks with known malicious substructures, enabling measurements of "repair completeness" and performance overhead. No mainstream resource yet includes detailed annotations for structural backdoors. Current benchmarks rarely address multi-branch gating, compiler-level infiltration [13], or AutoML-based Trojan designs [55, 56]. Classification tasks (e.g., *CIFAR-10*, *ImageNet*) dominate, overlooking more complex applications like segmentation or high-performance computing (HPC)-scale pipelines. Even fewer consider "post-training insertion" scenarios where malicious logic is introduced into ONNX files after the model is published. Bridging legacy data/weight sets with newly introduced structural infiltration could help the community assess repair strategies more rigorously.

6.3 Proposed Benchmarking Solutions

To move from ad-hoc structural demos to rigorous evaluation, the community needs purpose-built benchmarks addressing multi-branch gating, compiler-level infiltration, and supply-chain reinfection. Table 9 outlines four potential directions:

- **Extend existing suites (*TrojAI, BackdoorBench*):** Incorporate multi-branch, compiler-time, and large-scale LLM tasks using established scoring pipelines.
- **Repair benchmarks:** Provide ground-truth subgraph labels, enabling fair comparison of excision, unlearning, or distillation methods.
- **Arms-race tracks:** Host periodic competitions that evolve attacks and defenses, preventing over-fitting to static scenarios.
- **Industry & hub collaboration:** Aggregate suspicious uploads from real-world repositories. Overcome legal/licensing barriers to build a comprehensive, publicly accessible dataset.

Table 9. Roadmap for next-generation benchmarks on architectural backdoors.

Initiative	**Scope & Expected Gain**	**Key Obstacles**
Expand legacy suites (TrojAI, BackdoorBench)	Add multi-branch models, compiler inserts, LLM or multi-modal tasks	New generation scripts; licensing of non-open IRs.
Repair benchmarks	Ground-truth subgraph masks for subgraph-excision methods	Annotating malicious ops at scale; preserving benign performance.
Arms-race tracks	Annual events that evolve attacks/defenses	Organizer bandwidth; dynamic rules for success.
Industry & hub collaboration	Live scanning of user-submitted models	Privacy/legal issues; inconsistent licensing; incentives for submission.

Extensive expansion of backdoor benchmarks is needed to address architectural backdoors. Table 8 shows that recognized programs like *TrojAI* and *BackdoorBench* devote minimal attention to multi-branch or compiler-time triggers. Although a few works address LLM or multi-modal vulnerabilities, none provide large-scale, community-driven benchmarks. An adaptive approach that updates structural infiltration scenarios regularly, coupled with academic-industry collaboration on real repository scans, would keep defenders aligned with evolving adversarial techniques. We thus call for dedicated tasks, or expansions within popular frameworks, focusing on multi-branch gating, compiler-level insertions, and domain-shift reactivation, ensuring defenses are tested against the full spectrum of architectural backdoors.

7 Open Challenges and Future Directions

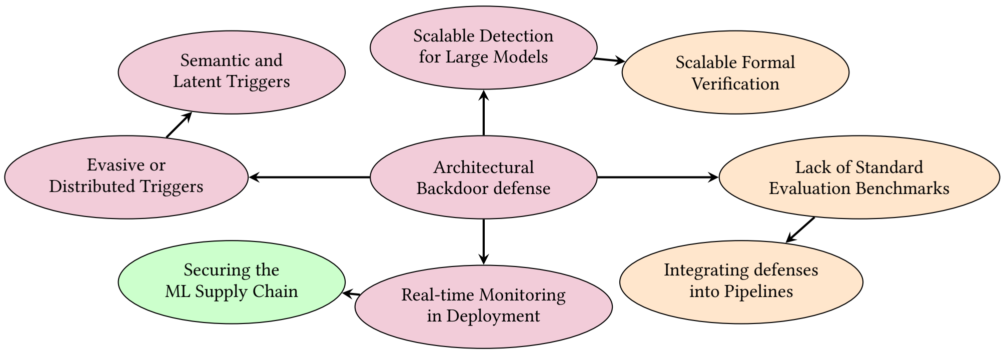
Semantic and Latent Triggers
Scalable Detection for Large Models
Scalable Formal Verification
Evasive or Distributed Triggers
Architectural Backdoor defense
Lack of Standard Evaluation Benchmarks
Securing the ML Supply Chain
Real-time Monitoring in Deployment
Integrating defenses into Pipelines

Fig. 13. **Conceptual map of open challenges in defending against architectural backdoors.**

Architectural backdoors present threats that conventional data-poisoning or parameter-centric defenses rarely address, especially when embedded at the compiler or hardware level. The preceding sections (§4–§6) have analyzed subgraph modifications, AutoML vulnerabilities, and the limited scope of current benchmarks. Despite these advances, several obstacles remain before comprehensive security against architectural backdoors can be achieved, particularly at scale. This section highlights the open challenges, then proposes directions for large-scale verification frameworks, supply-chain governance, and adaptive benchmarking, ultimately aiming for robust, end-to-end solutions that anticipate emerging infiltration tactics. Figure 13 summarizes how the major research and operational gaps inter-relate; the subsections that follow unpack each edge of the diagram in depth.

Summary of open challenges and future directions:

(1) **Scaling Formal Methods and Large-Scale Verification:** Satisfiability Modulo Theories (SMT)-based and abstract-interpretation verifiers achieve promising guarantees on feed-forward models of tens of millions of parameters (10^7), but attempts to push them to Transformer-scale routinely time-out or blow up in memory [60]. *How can compositional, approximate, or statistical verification frameworks certify safety properties of trillion-parameter or HPC-scale architectures without prohibitive cost?*

(2) **Addressing Multi-Path, Distributed, or Obfuscated Triggers:** Hidden backdoor logic is increasingly split across multiple attention heads, residual branches, or even modalities; triggers may require specific chain-of-thought (CoT), token sequences, or coordinated text–image pairs. Recent studies into distributed triggers [37], frequency-obfuscated vision backdoors (*LADDER* [45]), CoT hijacks such as *BadChain* [80], *DarkMind* [27], *ShadowCoT* [86], *TransTroj* fine-tuning persistence [74], and the open-source *BackdoorLLM* pipeline [40], achieve >90% attack success while bypassing current detectors, and no public benchmark yet exercises these cross-modal gates [87]. *What graph-analytic or constraint-solving techniques can reason about distributed activation patterns and reliably surface stealthy multi-branch trigger combinations?*

(3) **NAS and AutoML Vulnerabilities:** NAS and other AutoML pipelines can inadvertently embed, or deliberately plant, malicious sub-structures that optimize for a poisoned reward signal. *How can integrity checks on reward functions, search spaces, and validation loops guarantee that generated architectures are provably free of covert backdoor logic?*

(4) **Compiler/Hardware Synergy and Post-Training Backdoor Re-Introduction:** Malicious compilers or hardware Trojans can resurrect excised subgraphs after software-level cleansing via post-training reinsertion (e.g., *Shadow Logic* [29]) or IR manipulation, recreating backdoors at deploy time [13]. *Can we design end-to-end trusted toolchains that preserve a verifiable equivalence between audited source graphs and the binaries and silicon ultimately deployed?*

(5) **Adaptive Benchmarks and Continuous Updating:** Static challenge suites quickly become outdated as adversaries evolve. Preliminary hub-scanning tools such as *CLIBE* (deteCting NLP dynamIc Backdoor TransformEr) [82] remain proprietary, and no open governing body curates an evolving benchmark. *What incentive structures would support a living benchmark, including real-world corpus tracks and LLM-orchestrated defender baselines [7], that stay synchronized with emerging attack strategies?*

(6) **Specialized Architectures: Spiking Neural Networks (SNNs) and Visual State-Space Models (VSSMs):** Non-canonical computing graphs diverge from CNN/Transformer assumptions, creating blind spots for existing defenses. *Which abstraction layers or surrogate models will let us reason about temporally-coded or state-space activations without sacrificing threat coverage?*

(7) **Supply-Chain Governance, Policy, and Multi-Sector Collaboration:** Public hubs and proprietary exchanges lack harmonized auditing standards, giving attackers ample surface for injection. Successful large-scale defense will likely require joint policy, open-source tooling, and certification programs akin to software bill of material (SBOM) initiatives. *What regulatory, policy, and technical levers can align incentives across academia, industry, and government to secure the global model supply chain?*

Recent empirical work on GPT-style models by Miah and Bi [49], together with the comprehensive LLM survey by Zhao *et al.* [87], confirms that architectural backdoors have already migrated to billion-parameter language models.

7.1 Scaling Formal Methods and Large-Scale Verification

Problem statement: Formal, symbolic, and abstract-interpretation–based verifiers can certify small neural networks against backdoor behavior under bounded perturbations[17, 22, 32, 67]. Beyond roughly 10^7 parameters they either time-out or excessively over-approximate, leaving modern CNNs, vision–language transformers, and all contemporary LLMs effectively unverifiable. Compiler-time backdoor re-insertion, demonstrated by *ImpNet*/*Shadow Logic* attacks[13], further undercuts proof soundness: a source graph proven clean can still be corrupted during lowering to inference binaries.

Why current defenses fall short

- *State-space explosion.* Parameter counts scale super-linearly with depth and width; existing SMT or Mixed Integer Linear Programming (MILP) encodings become intractable for trillion-parameter models or multi-branch architectures with distributed gating triggers.
- *Compiler/Hardware gap.* Proofs apply to the abstract graph prior to optimization passes; malicious compilers or rogue hardware IP can violate the verified property post-compile, creating an unmonitored attack surface.

- *Lack of incremental benchmarks.* Public challenge suites rarely include models $>$ 100M parameters, so progress on scaling techniques is neither tracked nor rewarded.

Promising research directions

- *Compositional proofs.* Decompose networks into verifiable blocks (layers, residual paths) and stitch guarantees via assume–guarantee reasoning [59] or category-theoretic lenses [20]. Partial coverage can still block many distributed trigger designs.
- *HPC-assisted verification.* Cluster-scale symbolic execution frameworks, coupled with aggressive post-pruning (§5.1), could deliver tractable yet conservative proofs for very large models.
- *Runtime attestation.* Lightweight on-device monitors can hash or attest critical gating operations during inference, providing continuous assurance even if full compile-time proofs are infeasible.
- *Trusted toolchains.* End-to-end pipelines that cryptographically link a verified source graph to emitted binaries and hardware bitstreams would close the compiler gap; even partial formal checks retain value when supply chains are otherwise trusted.
- *Hierarchical IR differencing and proof-carrying code.* Coupling static taint analysis (e.g. *Batch Isolation Checker*'s Monoid-based proof [36]) with lowering passes can certify non-interference at compile time.

Bridging HPC pipelines with rigorous proofs thus remains a grand challenge, one that will require tight collaboration among ML-security specialists, compiler/toolchain engineers, and formal-methods researchers.

7.2 Addressing Multi-Path, Distributed, or Obfuscated Triggers

Problem statement: Modern backdoor designers increasingly distribute trigger logic across multiple computational paths, modalities, or time steps. CoT prompts, hidden gating sub-layers, and cross-modal token–image combinations can all serve as stealthy activation keys that no single neuron, branch, or dataset sample exposes [87]. Benchmark suites remain heavily skewed toward single-branch image classifiers, leaving multimodal and multi-path threats under-represented.

Why current defenses fall short

- *Isolation bias.* Structural detectors score each subgraph independently and thus miss backdoors that only activate when several branches co-fire in concert.
- *Evasive distributed triggers.* Distributed-gating networks [6, 37] and *LADDER's* frequency-domain vision triggers [45] adapt to pruning and still reach $>90\%$ attack success while keeping clean accuracy intact.
- *CoT hijacks withstand consistency checks.* Lightweight tuning pipelines such as *BadChain* [80], *DarkMind* [27], and *ShadowCoT* [86] embed triggers deep in the reasoning pathway, defeating feature-similarity detectors such as *Neural Cleanse* [73] and *MNTD* [35].
- *Modal gap.* No public benchmark yet evaluates triggers that fuse text with image or audio modalities, leaving multi-modal gates untested [72, 82].
- *Batch-context leaks.* A hidden path can pass information across examples inside one inference batch, enabling data theft and peer-output manipulation [36].

Promising research directions

- *Graph-level constraint solving.* Combine static dataflow graphs with dynamic activation traces to enumerate improbable multi-branch co-activations; graph neural networks (GNNs) could score subgraph tuples rather than single nodes.
- *Composite fuzzing for multimodal pipelines.* Iteratively synthesize paired inputs (e.g., prompt-image) that search the joint space for trigger conditions; coverage-guided feedback can steer toward rare co-activation states.
- *Robust multi-branch mitigation.* Adversarial unlearning or mask-and-fine-tune loops must prune all partial triggers. Layer-wise causal tracing, combined with sparsity-aware retraining (§5.1), may disable distributed gates without catastrophic accuracy loss.
- *Evolving benchmarks.* A living "arms-race track" that adds the latest multi-modal backdoor designs each release cycle would provide a moving target for defenders and avoid the staleness that plagues current benchmark corpora.

Table 10. **Benchmark coverage and key gaps**

Benchmark	**Included**	**Missing (Critical Gaps)**
TrojAI (IARPA)	Many poisoned & weight-backdoored models; standardized detection metrics	No multi-branch or compiler-level attacks; primarily data- and weight-focused
BackdoorBench	Unified framework for comparing defenses on standard image datasets; basic single-layer structural triggers	No compiler-tampering scenarios; lacks complex multi-trigger architectures; primarily designed around visible triggers
NeurIPS Trojan Detection Challenges	Primarily data- and parameter-based triggers in competition settings	No explicit inclusion of structural or compiler-inserted scenarios
BackdoorLLM (2024)	Prompt- and fine-tune-based LLM backdoors; open-source evaluation scripts	No structural/graph or compiler-level attacks; structural attacks: None
TrojanZoo (2022)	Static image datasets and code for backdoor research; easy defence benchmarking	No dynamic updates; no compiler-time or multi-branch attacks; structural attacks: None

7.3 NAS and AutoML Vulnerabilities

Problem statement. NAS and broader AutoML pipelines can embed backdoor logic if an attacker tampers with the reward function, the search code, or the compilation stage. Because these systems often output irregular, massive graphs, hidden submodules may be indistinguishable from legitimate skip connections [55, 56].

Why current defenses fall short

- *Opaque search traces.* Commercial NAS services rarely provide a verifiable log of evaluated candidates, so reviewers cannot tell whether a suspicious gating op was present during architecture selection.
- *Reward-function poisoning.* An adversary can bias the search toward architectures that conceal malicious subgraphs while still meeting published accuracy targets.
- *Compiler-time drift.* Even if the final high-level graph looks clean, low-level IR (e.g., ONNX) can be patched post-training, re-introducing hidden branches before deployment.
- *Scale barrier.* Static scanners struggle with the tens of thousands of candidate graphs produced in a single large-scale search, making exhaustive auditing impractical.

Promising research directions

- *Cryptographically verifiable search logs.* Hash-chained records of every candidate (graph, seed, reward) would let third parties confirm that the deployed model derives from an untampered search trace.
- *IR differencing across compilation steps.* Automated checks can diff intermediate representations against the audited source graph, flagging any new ops inserted after the AutoML phase.
- *White-list–based architecture validation.* Require search controllers to declare an explicit list of allowable modules; any unexplained layer (e.g., unconventional gating) fails certification.
- *Search-time anomaly detection.* Lightweight graph fingerprints computed during NAS can spot sudden structural deviations, catching backdoor candidates before expensive training finishes.

These measures should eventually align with supply-chain hardening (see §5.5) so that a verified AutoML output is not silently altered by later compilation or hardware integration.

7.4 Compiler/Hardware Synergy and Post-Training Backdoor Re-Introduction

Problem statement. Even after a model passes structural scans, a malicious compiler or bitstream generator can re-insert excised subgraphs, and hardware Trojans can activate dormant gates at runtime [75]. Recent work such as *ImpNet* [13] and *Shadow Logic* [29] modify IR by only a few hundred bytes, silently restoring the backdoor. *Batch Isolation Checker* further shows that ONNX-level graph edits can inject a backdoor long after training, enabling within-batch data leakage and output manipulation [36]. Once the binary or FPGA bitstream is shipped, software-level defenses have no visibility.

Why current defenses fall short

- *Visibility gap.* Audits stop at the high-level graph; they rarely inspect the lowered IR or microcode, where a single fused op can hide a trigger.
- *Trusting-trust scenario.* If the entire toolchain is compromised, recompiling from source reproduces the backdoor unless the compiler itself is verified or multi-compiled.
- *Scalability of formal proofs.* Current verifiers cannot handle billion-parameter graphs and post-compile inserts; source-level proofs are invalidated by later IR rewrites [60].
- *Hardware opacity.* Bitstreams for ASICs/FPGAs are proprietary; defenders cannot easily map binary regions back to functional blocks, let alone prove the absence of stealth logic [75].

Promising research directions

- *Reproducible and signed builds.* Hash-chained, deterministic compilation (e.g., Bazel's `--reproducible` flag [5] and its CI caching study [88] or container capture via *ReproZip* [10]) and diverse double-compiling [76] lets verifiers confirm that the deployed binary matches an audited source graph.
- *Hierarchical IR differencing and proof-carrying code.* Each lowering pass attaches a machine-checkable proof that no new control-flow edges reach the backdoor label set; partial proofs still add value inside a trusted supply chain.
- *Bitstream attestation and runtime monitors.* Split the accelerator fabric into whitelisted regions and apply side-channel guards or runtime hash checks for unauthorized activations [65].
- *Cross-layer certification.* Unify software and hardware verification by expressing both the neural graph and the accelerator netlist in a single SMT-based meta-model, enabling end-to-end equivalence checks.

Bridging compiler transformations, hardware design, and ML graphs therefore demands a holistic, supply-chain view; defensively chaining even partial formal checks can raise the bar until tool-support scales to full proofs.

7.5 Adaptive Benchmarks and Continuous Updating

Problem statement. Most public backdoor testbeds such as *TrojAI*, *BackdoorBench*, *TrojanZoo*, and the language-model-focused *BackdoorLLM* suite [40, 58, 78], remain largely static once released. Consequently, defenses tuned to a single round of pixel or weight triggers can appear robust while collapsing against hidden-graph or compiler-time attacks that the benchmark never covered.
Why current defenses fall short

- *Stale attack sets.* Few benchmarks introduce multi-modal or compiler-injected triggers, leaving entire threat classes unmeasured.
- *Real-world drift.* The Guardian scanner recently flagged 57 suspect ONNX graphs on Hugging Face [64]; none of those samples resemble existing challenge rounds, exposing a realism gap.
- *HPC compilation loops.* Production pipelines re-optimize models on every deployment. A defense that succeeds on a frozen weight file may fail after even one hardware-aware recompilation.

Promising research directions

- *Annual "arms-race" tracks.* Each round injects *new* infiltration vectors (stealth gating, text–image triggers, post-training IR rewrites), forcing defenses to generalize beyond last year's tactics.
- *HPC-pipeline simulation.* Benchmarks should recompile each submitted model through multiple optimization passes, rewarding detectors that remain stable across binary drift.
- *Rolling real-world corpus tracks.* Publicly harvested incident sets (e.g. the PAIT-ONNX and PAIT-TF reports from Protect AI [64] plus the 269 leaking models found by Küchler *et al.*'s scan of 1,680 ONNX graphs [36]) can supply continually refreshed challenge seeds that mirror live supply-chain threats.
- *Live scoreboards and gap analytics.* A web dashboard summarizing which defenses block which vectors will highlight lingering blind spots and channel research effort.
- *Multi-modal extensions.* Future rounds must include text–image or audio–text gates that no current benchmark yet exercises, as highlighted by Zhao *et al.*'s LLM survey [87].
- *LLM-orchestrated defender baselines.* Recent work shows that large language models can coordinate autonomous cyber-defense actions [7]; incorporating such agents as detectors/repairers would test whether benchmarks keep pace with AI-driven defense.

A continuously evolving benchmark ecosystem, anchored by annual arms-race tracks, realistic corpus feeds, and HPC-aware recompilation loops, offers the best hope of preventing defenses from over-fitting to yesterday's attacks.

7.6 Specialized Architectures: Spiking Neural Networks and Visual State-Space Models

Problem statement. Architectural backdoors are no longer confined to convolutional or Transformer families. Spiking Neural Networks (SNNs) and Visual State-Space Models (VSSMs) have both been shown vulnerable to stealthy, architecture-level attacks: Sneaky Spikes embeds persistent triggers in neuromorphic timing channels [1], while BadScan tampers with VSSM state-update blocks [14]. As these models proliferate in robotics, real-time control, and neuromorphic hardware, ignoring their unique threat surface leaves an ever-growing security gap.
Why current defenses fall short

- *Continuous-activation bias.* Most detectors assume real-valued activations; they miss micro-timing or spike-rate perturbations that encode SNN backdoors.
- *Hidden recurrent pathways.* VSSMs update an internal state through learned linear systems, so a malicious matrix can stay dormant until a specific input trajectory occurs, evading static graph scans.

- *Lack of benchmark.* No public challenge currently includes spiking or state-space triggers; defenders have no ground truth for evaluation.
- *Tool-chain gap.* Neuromorphic compilers and on-device learning loops lack the signing and reproducibility features now standard in mainstream ML toolchains (§7.4).

Promising research directions

- *Event-level tracing and invariants.* Define spike-timing windows or state-flow invariants and flag inputs that violate them.
- *Domain-specific benchmarks.* Extend *TrojAI* [72] or *BackdoorBench* [78] with SNN timing triggers and VSSM state-matrix inserts, using an "arms-race" update cadence (§7.5).
- *Formal abstractions for spikes and states.* Adapt mixed-signal verification or control-theoretic reachability analysis to reason about timing-encoded or state-encoded backdoors.
- *Cross-disciplinary* Combine neuromorphic-hardware provenance checks with the reproducible-build approach already proposed for standard accelerators (§7.4).

7.7 Supply-Chain Governance, Policy, and Multi-Sector Collaboration

Problem statement: Technical countermeasures are necessary but insufficient: architectural backdoors threaten regulated sectors, healthcare, finance, autonomous vehicles, where liability, compliance, and public safety considerations dominate [77]. Without verifiable provenance and enforceable policy, a single compromised compiler stage can undermine every software-level defense.

Why current defenses fall short

- *Fragmented standards.* Cryptographic signing, reproducible builds, and audit logging exist, but no cross-industry baseline mandates them for neural-network toolchains.
- *Unclear liability.* When hidden logic is discovered, it is ambiguous whether fault lies with the model provider, compiler vendor, or third-party library maintainer.
- *Regulatory lag.* Emerging laws (e.g., EU AI Act 2024 [18]) require risk documentation but stop short of specifying architecture-integrity attestation.
- *Domain-specific blind spots.* Safety-critical systems rarely perform "white-box" architectural audits, focusing instead on data or post-hoc performance metrics.

Promising research and policy directions

- *Signed, reproducible toolchains.* Mandate hash-chained logs and deterministic builds for each compile step, aligning with NIST AI RMF 1.0 guidance and ISO/IEC 42001 proposals [52].
- *Sector-specific integrity audits.* Regulators could require provable "white-box" scans of autonomous-vehicle or medical models before type approval, similar to functional-safety testing today.
- *Shared liability frameworks.* Clarify contractual obligations so that model providers, tool-vendors, and cloud operators share responsibility for preventing compiler-level backdoors.
- *Third-party certification.* Create accredited services that certify neural architectures against malicious subgraph insertion, mirroring SOC 2 or Common Criteria for traditional software.
- *Cross-sector working groups.* Foster collaboration among neuromorphic, HPC, and policy communities to translate supply-chain lessons from conventional software into the ML domain.

Embedding these governance layers into the ML lifecycle ensures that technical defenses such as the reproducible-build pipeline in §7.4, are backed by enforceable policy, shifting architectural-backdoor mitigation from best-effort practice to industry norm.

7.8 Proposed Research Roadmap

Table 11. **Short-Term vs. Long-Term Research Goals for Architectural Backdoor Security**

Challenge	Short-Term Goals	Long-Term Goals
Large-Scale Verification	Develop HPC-friendly partial proofs; compositional checks on sub-networks	Full crosslayer verification that scales to trillion-parameter LLMs or HPC pipelines
Multi-Path, Distributed Triggers	Integrate multi-branch infiltration into existing detection tools; refine gating instrumentation	Deploy constraint-based or real-time gating checks in industrial systems; unify with formal proofs of activation patterns
Malicious NAS/AutoML	Cryptographically log each architecture search step; IR-level differencing	Fully transparent AutoML pipelines with forced architecture declarations and standardized subgraph scanning
Compiler/Hardware Backdoor Re-Introduction	Expand reproducible builds; cryptographic signing at compile time	End-to-end synergy across compiler, hardware bitstreams, and ML frameworks; hardware-level concurrency checks
Adaptive Benchmarks	Add pilot structural infiltration to TrojAI [72] or BackdoorBench [78]	Establish continuous, arms-race style expansions for HPC or multi-modal infiltration
Supply-Chain Governance and Policy	Introduce code signing and domain-specific compliance checks	Formal policy frameworks ensuring secure pipelines from architecture code to final hardware implementations

Table 11 explicitly outlines actionable short-term goals and ambitious long-term objectives to guide researchers, practitioners, and policymakers. For instance, benchmark maintainers should introduce at least one multi-branch or compiler-inserted Trojan scenario in the next round of *TrojAI* or *BackdoorBench*. For longer-term grand challenges, such as full cross-layer verification of trillion-parameter models, new theoretical frameworks may be required that constrain model architecture to enable tractable verification without performance degradation. Such challenges necessitate broad, interdisciplinary collaboration. By coordinating short-term efforts (e.g., cryptographic logs, partial compositional proofs, pilot structural infiltration in existing benchmarks) with these long-term objectives (full HPC-scale verification, continuous pipeline scanning, multi-modal backdoor expansions), the field can gradually construct a robust, end-to-end security strategy.

7.9 Summary

Architectural backdoors, ranging from multi-path triggers to NAS-biased topologies, expose gaps in parameter-centric defences; priorities now are (i) scalable formal proofs for multi-branch HPC models, (ii) transparent and logged AutoML/compilation pipelines, (iii) hardened supply chains with reproducible builds, (iv) adaptive benchmarks that evolve with new infiltration patterns, and (v) extending tests to LLM and multi-modal systems. Ultimately, progress in these open challenges can raise the bar for architectural backdoor security across software and hardware boundaries. The community-encompassing ML security researchers, compiler engineers, hardware designers, and policy/regulatory experts-must collaborate to build end-to-end, verified pipelines that deter architectural backdoors even in large-scale, rapidly evolving AI ecosystems. By integrating supply-chain assurance, HPC-scale verification, and iterative benchmark expansions, we can move closer to a future where malicious subgraphs, multi-branch gating, and post-training backdoor reintroductions are consistently detected and neutralized in practice.

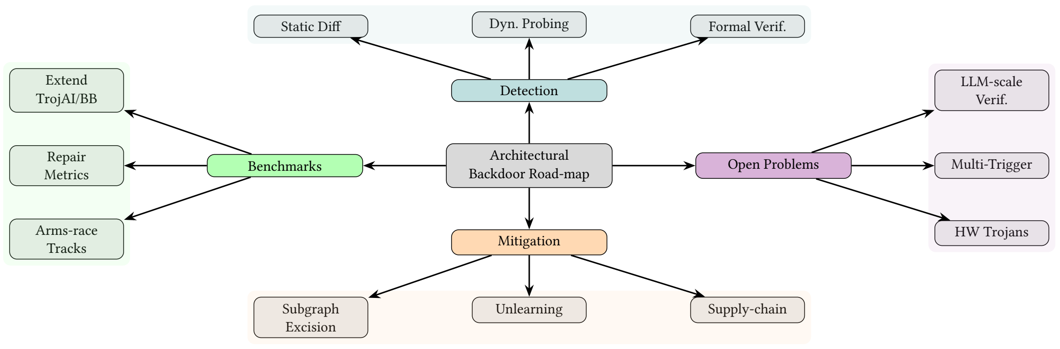

Fig. 14. **Research roadmap for architectural-backdoor security.**The figure groups open work into four color-banded themes: teal = *detection*, orange = *mitigation*, green = *benchmarking*, and violet = *open problems* [18, 27, 65].

8 Conclusion and Outlook

Figure 14 groups the remaining research challenges into four color-coded themes: detection, mitigation, benchmarking, and open problems which correspond to successive stages in the machine-learning supply chain. The discussion below walks clockwise through those bands.

Detection (teal band): Static differencing, dynamic probing, and formal verification form the first checkpoint. Yet multi-branch triggers such as Langford's distributed gates[37], *DarkMind's* latent CoT hijacks [27], and *BadChain's* lightweight tuning [80] routinely evade single-path heuristics. SMT-based provers stall on billion-parameter graphs [60], and compiler-time insertions like *Shadow Logic* [29] compromise even "clean" source graphs. Scalable, graph-wide reasoning therefore remains open.

Mitigation (orange band): Sub-graph excision, unlearning, or signed tool-chains can lower attack success, but a poisoned build system can silently re-insert the backdoor (*ImpNet* [13]) or activate a dormant hardware Trojan [65]. Supply-chain hardening, hash-chained, reproducible builds, container capture, and diverse double-compiling [76], bind verified graphs to emitted binaries and close the "trusting-trust" loop.

Benchmarking (green band): Legacy challenges *(TrojAI, BackdoorBench, TrojanZoo)* seldom test structural attacks, and no round yet stresses cross-modal gates highlighted by Zhao *et al.* [87]. Hub scanners such as *CLIBE* model [82] and the PAIT corpus [64] reveal stealthy exploits already in the wild. A rolling, arms-race track, augmented with explicit repair metrics to grade mitigation quality, would turn benchmarks into a living barometer rather than a static checklist.

Open problems (violet band): LLM-scale verification, multi-trigger reasoning, and hardware–compiler synergy are unresolved frontiers. The EU AI Act [18] and NIST AI RMF 1.0 [52] require provenance but still lack architecture-integrity clauses. Consensus across research, industry, and policy is needed before HPC and multi-modal deployments become the default.

8.1 Call to action

- *Detect*: build graph-level solvers that scale to trillion-parameter LLMs and expose multi-branch activations.
- *Mitigate*: embed reproducible builds, IR differencing, and runtime attestation in CI/CD pipelines.
- *Benchmark*: launch yearly arms-race rounds, scored on detection and repair.

- *Govern*: codify architectural-integrity attestation and shared liability in emerging AI standards and policy.

Architectural backdoor defense, therefore, spans design, compilation, and deployment, and succeeds only through joint effort of data scientists, software engineers, hardware designers, security experts, and regulators acting in concerted collaboration. As HPC-scale deep learning accelerates, closing these structural loopholes now is pivotal to safeguarding future AI systems. We hope this survey catalyzes that multi-disciplinary collaboration.

References

[1] Gorka Abad, Oguzhan Ersoy, Stjepan Picek, and Aitor Urbieta. 2024. Sneaky Spikes: Uncovering Stealthy Backdoor Attacks in Spiking Neural Networks with Neuromorphic Data. In *Proceedings of the 2024 Network and Distributed System Security (NDSS) Symposium*. Internet Society, San Diego, CA, USA, 1–20. doi:10.14722/ndss.2024.24334

[2] Protect AI. 2025. Six Months of Guardian: 4.47 Million Models Scanned on Hugging Face. https://protectai.com/blog/hugging-face-protect-ai-six-months-in. Accessed 11 Jun 2025.

[3] Yang Bai, Gaojie Xing, Hongyan Wu, Zhihong Rao, Chuan Ma, Shiping Wang, Xiaolei Liu, Yimin Zhou, Jiajia Tang, Kaijun Huang, and Jiale Kang. 2025. Backdoor Attack and Defense on Deep Learning: A Survey. *IEEE Transactions on Computational Social Systems* 12, 1 (2025), 404–434. doi:10.1109/TCSS.2024.3482723

[4] Peter Bajcsy and Maxime Bros. 2024. Interactive Simulations of Backdoors in Neural Networks. arXiv:2405.13217. https://arxiv.org/abs/2405.13217 National Institute of Standards and Technology.

[5] Bazel Project. 2024. Bazel: Reproducible Builds. https://bazel.build/docs/user-manual#reproducible-builds. Online; accessed 22 Apr 2024.

[6] Mikel Bober-Irizar, Ilia Shumailov, Yiren Zhao, Robert Mullins, and Nicolas Papernot. 2023. Architectural Backdoors in Neural Networks. In *Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 24595–24604. doi:10.1109/CVPR52729.2023.02356

[7] Sebastián R. Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, and Alvaro A. Cardenas. 2025. Large Language Models are Autonomous Cyber Defenders. In *Proceedings of the IEEE Conference on Artificial Intelligence (CAI) Workshop on Adaptive Cyber Defense*. IEEE, Santa Clara, CA, USA, 1–18. https://arxiv.org/abs/2505.04843

[8] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2019. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *Proceedings of the AAAI Workshop on Artificial Intelligence Safety (SafeAI 2019) co-located with the Thirty-Third AAAI Conference on Artificial Intelligence (CEUR Workshop Proceedings, Vol. 2301)*. CEUR-WS.org, Honolulu, HI, USA, 66–73. https://ceur-ws.org/Vol-2301/paper_18.pdf

[9] Wei-Jie Chen, Zhi-Hao Li, and Min Zhang. 2025. HGBA: Heterogeneous Graph Backdoor Attack via Relation-Aware Triggers. 21 pages. https://www.arxiv.org/abs/2506.00191

[10] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. ReproZip: Computational Reproducibility With Ease. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, San Francisco, CA, USA, 2085–2088. doi:10.1145/2882903.2899401

[11] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. 2020. SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems. In *Proceedings of the 2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, San Francisco, CA, USA, 48–54. doi:10.1109/SPW50608.2020.00025

[12] Sheng-Yen Chou, Pin-Yu Chen, and Tsung-Yi Ho. 2023. How to Backdoor Diffusion Models?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE / CVF, Vancouver, BC, Canada, 4015–4024. doi:10.1109/CVPR52729.2023.00391

[13] Eleanor Clifford, Ilia Shumailov, Yiren Zhao, Ross Anderson, and Robert Mullins. 2024. ImpNet: Imperceptible and Blackbox-Undetectable Backdoors in Compiled Neural Networks. In *Proceedings of the 2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE Computer Society, Los Alamitos, CA, USA, 344–357. doi:10.1109/SaTML59370.2024.00024

[14] Om Suhas Deshmukh, Sankalp Nagaonkar, Achyut Mani Tripathi, and Ashish Mishra. 2024. BadScan: An Architectural Backdoor Attack on Visual State Space Models. *arXiv* abs/2411.17283 (2024), 11 pages. https://arxiv.org/abs/2411.17283 Preprint.

[15] Andis Draguns, Andrew Gritsevskiy, Sumeet Ramesh Motwani, and Christian Schroeder de Witt. 2024. Unelicitable Backdoors in Language Models via Cryptographic Transformer Circuits. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*. Curran Associates, Inc., Vancouver, BC, Canada, 1–26. https://proceedings.neurips.cc/paper_files/paper/2024/file/6087a60306544be7ba0d0cf34aa93c8f-Paper-Conference.pdf Main Conference Track.

[16] Jacob Dumford and Walter J. Scheirer. 2020. Backdooring Convolutional Neural Networks via Targeted Weight Perturbations. In *Proceedings of the 2020 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE, Houston, TX, USA, 1–9. doi:10.1109/IJCB48548.2020.9304875

[17] ETH SRI Lab. 2020. ERAN: ETH Robustness Analyzer for Neural Networks. https://github.com/eth-sri/eran. Online; accessed 28 Apr 2025.

[18] European Parliament and Council of the European Union. 2024. Regulation (EU) 2024/— on Harmonised Rules on Artificial Intelligence
(Artificial Intelligence Act) and Amending Certain Union Legislative Acts. Provisional consolidated text adopted 13 March 2024. https://data.consilium.europa.eu/doc/document/PE-44-2024-INIT/en/pdf Final citation in *Official Journal of the European Union* pending.

[19] Md Omar Faruque, Peter Jamieson, Ahmad Patooghy, and Abdel-Hameed A. Badawy. 2024. Unleashing GHOST: An LLM-Powered Framework for Automated Hardware Trojan Design. *arXiv* abs/2412.02816 (2024), 11 pages. https://arxiv.org/abs/2412.02816 Preprint.

[20] Brendan Fong, David I. Spivak, and Rémy Tuyéras. 2019. Backprop as Functor: A Compositional Perspective on Supervised Learning. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*. IEEE, Los Angeles, CA, USA, 1–13. doi:10.1109/LICS.2019.8785677

[21] Yuntao Gao, Chengyu Liu, Sanket Bhattad, Min Du, Xia Hu, and Jufeng Yang. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC '19)*. Association for Computing Machinery, San Juan, PR, USA, 113–125. doi:10.1145/3359789.3359790

[22] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P 2018)*. IEEE Computer Society, Los Alamitos, CA, USA, 3–18. doi:10.1109/SP.2018.00058

[23] Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. 2022. Planting Undetectable Backdoors in Machine Learning Models. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, Denver, CO, USA, 931–942. doi:10.1109/FOCS54457.2022.00089

[24] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv* abs/1708.06733 (2017), 14 pages. https://arxiv.org/abs/1708.06733 Preprint.

[25] Xihe Gu, Greg Fields, Yaman Jandali, Tara Javidi, and Farinaz Koushanfar. 2024. Trojan Cleansing with Neural Collapse. *arXiv* abs/2411.12914 (2024), 11 pages. https://arxiv.org/abs/2411.12914 Preprint.

[26] Chuan Guo, Ruihan Wu, and Kilian Q. Weinberger. 2021. On Hiding Neural Networks Inside Neural Networks. arXiv preprint arXiv:2002.10078, 14 pages. https://arxiv.org/abs/2002.10078 Version 3, May 2021.

[27] Zhen Guo and Reza Tourani. 2025. DarkMind: Latent Chain-of-Thought Backdoor in Customized LLMs. *arXiv* abs/2501.18617, 1 (2025), 1–16. arXiv:2501.18617 https://arxiv.org/abs/2501.18617 Preprint, 24 Jan 2025.

[28] Yingzhe He, Zhili Shen, Chang Xia, Jingyu Hua, Wei Tong, and Sheng Zhong. 2024. SGBA: A Stealthy Scapegoat Backdoor Attack against Deep Neural Networks. *Computers & Security* 136 (2024), 103523. doi:10.1016/j.cose.2023.103523

[29] HiddenLayer. 2024. ShadowLogic: Compiled Model Graph Manipulation for Persistent Backdoors. Technical Advisory Report. https://hiddenlayer.com/research/shadowlogic-backdoors-in-model-graphs

[30] IARPA and NIST. 2024. IARPA TrojAI Program — Round 15 Release. https://trojai.nist.gov/TrojAI_Round15.html. Accessed 2025-06-11.

[31] Hengrui Jia, Sierra Wyllie, Akram Bin Sediq, Ahmed Ibrahim, and Nicolas Papernot. 2025. Backdoor Detection through Replicated Execution of Outsourced Training. In *Proceedings of the 3rd IEEE Conference on Secure and Trustworthy Machine Learning (SaTML '25)*. IEEE, Copenhagen, Denmark, 20 pages. https://arxiv.org/abs/2504.00170

[32] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV) (LNCS, Vol. 10426)*. Springer, Heidelberg, Germany, 97–117. https://link.springer.com/chapter/10.1007/978-3-319-63387-9_5

[33] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2019. Marabou: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV)*. Springer, New York, NY, USA, 443–452.

[34] Dimitri Kokkonis, Michaël Marcozzi, Emilien Decoux, and Stefano Zacchiroli. 2025. ROSA: Finding Backdoors with Fuzzing . In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 720–720. https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00183

[35] Soheil Kolouri, Charles E. Rivera, Heiko Hoffmann, Pingfan Song, Farshad Khorrami, and Richard J. Radke. 2021. Meta Neural Trojan Detection. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 1–18. doi:10.1109/SP40001.2021.00034

[36] Nicolas Küchler, Ivan Petrov, Conrad Grobler, and Ilia Shumailov. 2025. Architectural Backdoors for Within-Batch Data Stealing and Model Inference Manipulation. arXiv:2505.18323 [cs.CR] arXiv:2505.18323, May 2025.

[37] Harry Langford, Ilia Shumailov, Yiren Zhao, Robert Mullins, and Nicolas Papernot. 2025. Architectural Neural Backdoors from First Principles. In *Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 60–60. https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00060

[38] Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. 2021. Backdoor Attacks on Pre-trained Models by Layerwise Weight Poisoning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021)*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 3023–3032. doi:10.18653/v1/2021.emnlp-main.241

[39] Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. 2021. Backdoor Attacks on Pre-trained Models by Layerwise Weight Poisoning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 3023–3032. doi:10.18653/v1/2021.emnlp-main.241

[40] Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. 2024. BackdoorLLM: A Comprehensive Benchmark for Backdoor Attacks on Large Language Models. *arXiv* abs/2408.12798 (2024), 20 pages. https://arxiv.org/abs/2408.12798 Preprint.

[41] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. 2022. Backdoor Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* 33, 12 (2022), 7475–7489. doi:10.1109/TNNLS.2022.3182979

[42] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. 2021. Anti-Backdoor Learning: Training Clean Models on Poisoned Data. In *Advances in Neural Information Processing Systems (NeurIPS '21, Vol. 34)*. Curran Associates, Inc., Virtual Event, 14900–14912. doi:10.5555/3540261.3541403

[43] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. 2021. Neural Attention Distillation: Erasing Backdoor Triggers from Deep Neural Networks. In *Proceedings of the 9th International Conference on Learning Representations (ICLR '21)*. OpenReview.net, Virtual Event (originally Addis Ababa, Ethiopia), 19 pages. https://openreview.net/forum?id=9l0K4OM-oXE Poster, ICLR 2021.

[44] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. 2021. Backdoor Attacks and Defenses in Deep Learning: A Survey. *IEEE Access* 9 (2021), 103 114. doi:10.1109/TCSS.2024.3482723

[45] Dazhuang Liu, Yanqi Qiao, Rui Wang, Kaitai Liang, and Georgios Smaragdakis. 2025. LADDER: Multi-Objective Backdoor Attack via Evolutionary Optimization. In *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS '25)*. Internet Society, San Diego, CA, USA, 18 pages. https://www.ndss-symposium.org/wp-content/uploads/2025-1061-paper.pdf Article-style paper, NDSS 2025.

[46] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security '18)*. USENIX Association, Baltimore, MD, USA, 273–290. https://doi.org/10.1007/978-3-030-00470-5_13

[47] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, London, United Kingdom, 1265–1282. doi:10.1145/3319535.3363216

[48] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS 2018)*. Internet Society, San Diego, CA, USA, 1–15. doi:10.14722/ndss.2018.23291

[49] Abdullah Arafat Miah and Yu Bi. 2024. Exploiting the Vulnerability of Large Language Models via Defense-Aware Architectural Backdoor. *arXiv* abs/2409.01952 (2024), 16 pages. https://arxiv.org/abs/2409.01952 Preprint.

[50] Rui Min, Zeyu Qin, Nevin L. Zhang, Li Shen, and Minhao Cheng. 2024. Breaking the False Sense of Security in Backdoor Defense through Re-Activation Attack. In *Advances in Neural Information Processing Systems (NeurIPS '24)*. Curran Associates, Inc., Vancouver, BC, Canada, 37 pages. https://proceedings.neurips.cc/paper_files/paper/2024/file/d06537b4b38ccf008a54559d2c56fa23-Paper-Conference.pdf

[51] Rui Min, Zeyu Qin, Nevin L. Zhang, Li Shen, and Minhao Cheng. 2024. Uncovering, Explaining, and Mitigating the Superficial Safety of Backdoor Defense. In *Advances in Neural Information Processing Systems (NeurIPS '24)*. Curran Associates, Inc., Vancouver, BC, Canada, 28 pages. https://openreview.net/pdf?id=qZFshkbWDo

[52] National Institute of Standards and Technology. 2023. *Artificial Intelligence Risk Management Framework (AI RMF) 1.0*. Technical Report NIST AI 100-1. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf Official release, 26 January 2023.

[53] NeurIPS Trojan Detection Challenge Organizers. 2020. NeurIPS 2020 Competition Track: Trojan Detection Challenge. https://neurips.cc/Conferences/2020/CompetitionTrack#trojan-detection. Accessed 28 Apr 2025.

[54] Liangbo Ning, Wenqi Fan, and Qing Li. 2025. Exploring Backdoor Attack and Defense for LLM-empowered Recommendations. *arXiv* abs/2504.11182 (2025), 21 pages. https://arxiv.org/abs/2504.11182 Preprint.

[55] Ren Pang, Changjiang Li, Zhaohan Xi, Shouling Ji, and Ting Wang. 2022. Neural Architectural Backdoors. *arXiv preprint* arXiv:2210.12179 (Oct. 2022), 15 pages. doi:10.48550/arXiv.2210.12179 Version 2, revised 7 Nov 2022.

[56] Ren Pang, Changjiang Li, Zhaohan Xi, Shouling Ji, and Ting Wang. 2023. The Dark Side of AutoML: Towards Architectural Backdoor Search. In *Proceedings of the 11th International Conference on Learning Representations (ICLR '23)*. OpenReview.net, Kigali, Rwanda, 16 pages. https://openreview.net/forum?id=bsZULlDGXe Poster, ICLR 2023.

[57] Ren Pang, Zhaohan Xi, Shouling Ji, Xiapu Luo, and Ting Wang. 2022. On the Security Risks of AutoML. In *Proceedings of the 31st USENIX Security Symposium*. USENIX Association, Boston, MA, USA, 3953–3970. https://www.usenix.org/conference/usenixsecurity22/presentation/pang-ren

[58] Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, Xiapu Luo, and Ting Wang. 2022. TrojanZoo: Towards Unified, Holistic, and Practical Evaluation of Neural Backdoors. In *Proceedings of the 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P 2022)*. IEEE, Genoa, Italy, 684–702. doi:10.1109/EuroSP53844.2022.00048

[59] Corina S. Pasăreanu, Divya Gopinath, and Huafeng Yu. 2018. *Compositional Verification for Autonomous Systems with Deep Learning Components*. Technical Report arXiv:1810.08303. NASA Ames Research Center. https://arxiv.org/abs/1810.08303 Technical Report & arXiv pre-print.

[60] Long H. Pham and Jun Sun. 2022. Verifying Neural Networks Against Backdoor Attacks. In *Computer Aided Verification – 34th International Conference, CAV 2022 (Lecture Notes in Computer Science, Vol. 13371)*. Springer, Haifa, Israel, 171–192. doi:10.1007/978-3-031-13185-1_9

[61] Dorde Popovic, Amin Sadeghi, Ting Yu, Sanjay Chawla, and Issa Khalil. 2025. DeBackdoor: A Deductive Framework for Detecting Backdoor Attacks on Deep Models with Limited Data. *arXiv* abs/2503.21305 (2025), 20 pages. doi:10.48550/arXiv.2503.21305 Preprint.

[62] Xiangyu Qi, Tinghao Xie, Ruizhe Pan, Jifeng Zhu, Yong Yang, and Kai Bu. 2022. Towards Practical Deployment-Stage Backdoor Attack on Deep Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF, New Orleans, LA, USA, 14865–14875. doi:10.1109/CVPR52688.2022.01299

[63] Joseph Rance, Yiren Zhao, Ilia Shumailov, and Robert D. Mullins. 2023. Augmentation Backdoors. In *Proceedings of the ICLR 2023 Workshop on Backdoor Attacks and Defenses in Machine Learning (BANDS '23)*. OpenReview, Kigali, Rwanda (hybrid), 14 pages. https://openreview.net/forum?id=-CIOGGhkEfy Workshop paper; retrieved 2025-04-30.

[64] Protect AI Research. 2024. PAIT Threat Reports: ONNX and TensorFlow Architectural Backdoors Found in the Wild. https://protectai.com/insights/knowledge-base/backdoor-threats/PAIT-ONNX-200. Accessed May 2025.

[65] Anirban Sengupta, Aditya Anshul, Vishal Chourasia, and Nabendu Bhui. 2025. Security Vulnerability (Backdoor Trojan) During Machine Learning Accelerator Design Phases. *IT Professional* 27, 1 (2025), 65–72. doi:10.1109/MITP.2024.3519632

[66] Guangyu Shen, Siyuan Cheng, Zhuo Zhang, Guanhong Tao, Kaiyuan Zhang, Hanxi Guo, Lu Yan, Xiaolong Jin, Shengwei An, Shiqing Ma, and Xiangyu Zhang. 2025. BAIT: Large Language Model Backdoor Scanning by Inverting Attack Target. In *Proceedings of the 46th IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 1676–1694. https://www.computer.org/csdl/proceedings-article/sp/2025/223600a103/22K50yIvWta

[67] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An Abstract Domain for Certifying Neural Networks. In *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2019)*. Association for Computing Machinery, New York, NY, USA, 329–342. doi:10.1145/3290354

[68] 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 *Proceedings of the 46th IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 1620–1638. doi:10.1109/SP61157.2025.00161

[69] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A survey of hardware Trojan taxonomy and detection. *IEEE Design & Test of Computers* 27, 1 (2010), 10–25. doi:10.1109/MDT.2010.7

[70] Ken Thompson. 1984. Reflections on Trusting Trust. *Commun. ACM* 27, 8 (1984), 761–763. doi:10.1145/358198.358210

[71] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral Signatures in Backdoor Attacks. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Curran Associates, Inc., Red Hook, NY, USA, 8011–8021. https://dl.acm.org/doi/10.5555/3327757.3327896

[72] U.S. National Institute of Standards and Technology. 2025. TrojAI Program Homepage. https://pages.nist.gov/trojai. Last accessed 28 Apr 2025.

[73] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, Los Alamitos, CA, USA, 707–723. doi:10.1109/SP.2019.00031

[74] Hao Wang, Shangwei Guo, Jialing He, Hangcheng Liu, Tianwei Zhang, and Tao Xiang. 2025. Model Supply Chain Poisoning: Backdooring Pre-trained Models via Embedding Indistinguishability. In *Proceedings of the ACM Web Conference 2025 (TheWebConf '25)*. ACM, Sydney, Australia, 840–851. doi:10.1145/3696410.3714624 "TransTroj" attack — backdoors persist through downstream fine-tuning.

[75] Alexander Warnecke, Julian Speith, Jan-Niklas Moller, Konrad Rieck, and Christof Paar. 2024. Evil from Within: Machine Learning Backdoors Through Dormant Hardware Trojans . 906-922 pages. doi:10.1109/ACSAC63791.2024.00077

[76] David A. Wheeler. 2005. Countering Trusting Trust Through Diverse Double-Compiling (DDC). In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*. IEEE Computer Society, Los Alamitos, CA, USA, 13–26. http://dx.doi.org/10.1109/CSAC.2005.17

[77] World Economic Forum. 2025. Global Cybersecurity Outlook 2025. Insight Report. https://www.weforum.org/publications/global-cybersecurity-outlook-2025 Published 13 January 2025; highlights AI-driven supply-chain and backdoor threats.

[78] Baoyuan Wu, Hongrui Chen, Mingda Zhang, Zihao Zhu, Shaokui Wei, Danni Yuan, and Chao Shen. 2022. BackdoorBench: A Comprehensive Benchmark of Backdoor Learning. In *Proceedings of the NeurIPS 2022 Datasets and Benchmarks Track*. Curran Associates, Inc., Red Hook, NY, USA, 14 pages. https://openreview.net/pdf?id=31_U7n18gM7

[79] Jun Xia, Zhihao Yue, Yingbo Zhou, Zhiwei Ling, Yiyu Shi, Xian Wei, and Mingsong Chen. 2024. WaveAttack: Asymmetric Frequency Obfuscation-Based Backdoor Attacks Against Deep Neural Networks. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., Red Hook, NY, USA, 43549–43570. https://papers.nips.cc/paper_files/paper/2024/file/4ce18228ececb78bca04cbce069891b1-Paper-Conference.pdf

[80] Zhen Xiang, Fengqing Jiang, Zidi Xiong, Bhaskar Ramasubramanian, Radha Poovendran, and Bo Li. 2024. BadChain: Backdoor Chain-of-Thought Prompting for Large Language Models. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*. OpenReview.net, Vienna, Austria, Article c93SBwz1Ma, 28 pages. https://openreview.net/pdf?id=c93SBwz1Ma Code available at https://github.com/Django-Jiang/BadChain.

[81] Xiao Yang, Kai Zhou, Yuni Lai, and Gaolei Li. 2024. Defense-as-a-Service: Black-box Shielding against Backdoored Graph Models. https://arxiv.org/abs/2410.04916. arXiv:2410.04916 [cs.LG]; accessed 28 Apr 2025.

[82] Rui Zeng, Xi Chen, Yuwen Pu, Xuhong Zhang, Tianyu Du, and Shouling Ji. 2025. CLIBE: Detecting Dynamic Backdoors in Transformer-Based NLP Models. In *Proceedings of the 32nd Network and Distributed System Security Symposium (NDSS 2025)*. Internet Society, San Diego, CA, USA, 18 pages. doi:10.14722/ndss.2025.230478

[83] Yi Zeng, Si Chen, Won Park, Z. Morley Mao, Ming Jin, and Ruoxi Jia. 2022. Adversarial Unlearning of Backdoors via Implicit Hypergradient. In *Proceedings of the Tenth International Conference on Learning Representations (ICLR 2022)*. OpenReview.net, Virtual Conference, Article MeeQkFYVbzW, 28 pages. https://openreview.net/pdf?id=MeeQkFYVbzW

[84] Yi Zeng, Won Park, Z. Morley Mao, and Ruoxi Jia. 2021. Rethinking the Backdoor Attacks' Triggers: A Frequency Perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV 2021)*. IEEE, Los Alamitos, CA, USA, 2752–2761. https://openaccess.thecvf.com/content/ICCV2021/papers/Zeng_Rethinking_the_Backdoor_Attacks_Triggers_A_Frequency_Perspective_ICCV_2021_paper.pdf

[85] Yechao Zhang, Yuxuan Zhou, Tianyu Li, Minghui Li, Shengshan Hu, Wei Luo, and Leo Yu Zhang. 2025. Secure Transfer Learning: Training Clean Models Against Backdoor in Pre-Trained Encoder and Downstream Dataset. In *Proceedings of the 46th IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 1732–1750. https://www.computer.org/csdl/proceedings-article/sp/2025/223600b639/26hiUrJfICA

[86] Gejian Zhao, Hanzhou Wu, Xinpeng Zhang, and Athanasios V. Vasilakos. 2025. ShadowCoT: Cognitive Hijacking for Stealthy Reasoning Backdoors in Large Language Models. https://arxiv.org/abs/2504.05605. arXiv:2504.05605 [cs.CL]; accessed 28 Apr 2025.

[87] Shuai Zhao, Meihuizi Jia, Zhongliang Guo, Leilei Gan, Xiaoyu Xu, Xiaobao Wu, Jie Fu, Yichao Feng, Fengjun Pan, and Anh Tuan Luu. 2025. A Survey of Recent Backdoor Attacks and Defenses in Large Language Models. *Transactions on Machine Learning Research* 2025, Article 3527 (2025), 28 pages. https://openreview.net/pdf?id=wZLWuFHxt5 Accepted 12 Jan 2025; CC BY 4.0.

[88] Shenyu Zheng, Bram Adams, and Ahmed E. Hassan. 2024. Does Using Bazel Help Speed Up Continuous Integration Builds? *Empirical Software Engineering* 29, Article 110 (2024), 47 pages. doi:10.1007/s10664-024-10497-x

[89] Ruofan Zhu, Ganhao Chen, Wenbo Shen, Xiaofei Xie, and Rui Chang. 2025. My Model is Malware to You: Transforming AI Models into Malware by Abusing TensorFlow APIs. In *Proceedings of the 46th IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 449–466. https://www.computer.org/csdl/proceedings-article/sp/2025/223600a012/21B7Q4kpO7e