Prototype-Guided Defense in Edge AI: Securing Distributed Inference Against Model Poisoning in Resource-Constrained Systems

Fei Luo
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA.
fluo@oregonstate.edu

Bastian Steele
Department of Computer Science, University of Alabama at Birmingham, Birmingham, AL, USA.
steelebastian@uab.edu

Leif L. Johnston
Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV, USA.
lljohnston@unr.edu

Abstract

The proliferation of edge artificial intelligence has enabled distributed inference across heterogeneous, resource-constrained devices, yet this architectural shift introduces acute vulnerabilities to model poisoning attacks that corrupt local updates or inference outputs. Traditional defense mechanisms, often designed for centralized cloud environments, impose prohibitive computational and communication overheads when adapted to the periphery of the network. This paper presents a systemic framework for prototype-guided defense, a technique that leverages representative class prototypes to detect and mitigate anomalous parameter deviations during distributed inference. We argue that prototype learning offers a structurally lightweight, intrinsically interpretable mechanism for securing edge AI systems without demanding substantial memory, bandwidth, or energy budgets. The exposition covers the threat model of targeted model poisoning in split inference and federated edge settings, the architectural incorporation of prototype consistency checks into local inference nodes, and the resulting trade-offs among security, accuracy, latency, and resource consumption. Broader implications for governance, sustainability, and fairness are examined, including the risk of prototype bias amplification and the need for standardized verification protocols. Through cross-domain analysis spanning autonomous vehicles, smart healthcare, and industrial IoT, we demonstrate that prototype-guided defenses, when carefully calibrated, can significantly strengthen the resilience of edge AI deployments. The paper concludes with forward-looking recommendations for policy frameworks and open research directions that balance security guarantees with the inherent constraints of embedded systems.

Keywords

edge AI, model poisoning, prototype learning, distributed inference, resource-constrained systems, adversarial defense, system architecture.

1. Introduction

Edge artificial intelligence has emerged as a transformative paradigm for deploying machine learning models directly on devices at the network periphery, enabling low-latency inference, privacy preservation, and bandwidth efficiency [1]. Such systems, however, operate under severe resource limitations: limited memory, constrained processing power, intermittent connectivity, and stringent energy budgets. These constraints render conventional security mechanisms—such as heavy cryptographic verification, periodic global aggregation, or complex anomaly detection—impractical or even infeasible. Among the most insidious threats facing edge AI is model poisoning, where an adversary introduces malicious updates or corrupts inference outputs to degrade system performance or implant backdoors that activate under specific conditions [2].

Model poisoning differs from data poisoning in that it directly manipulates model parameters or gradient vectors rather than training data, making it particularly dangerous in collaborative inference scenarios such as split learning and federated edge networks [3]. In these architectures, local devices maintain partial models or feature extractors that are aggregated or transmitted to a central server; a single compromised node can propagate erroneous updates that perturb the global decision boundary. Existing defenses, including robust aggregation rules and differential privacy, often impose communication or computation costs that scale poorly with the number of participating devices and the dimensionality of model parameters [4]. Consequently, a critical gap exists in the literature for defense strategies that align with the operational realities of edge deployments.

This paper introduces prototype-guided defense as a principled approach to securing distributed inference against model poisoning in resource-constrained systems. Prototype learning, originally developed for interpretable and few-shot classification, constructs a set of class-representative vectors that encode the typical feature distribution of each category [5]. By enforcing that local model updates or intermediate representations remain consistent with these prototypes, the system can detect and reject anomalous contributions without requiring full model retraining or expensive cryptographic checks. The core insight is that prototypes serve as lightweight, memory-efficient reference anchors that can be stored and updated incrementally on each edge node, enabling decentralized anomaly detection with minimal overhead.

The remainder of this paper is organized as follows. Section 2 provides a detailed background on the threat model and the unique attack surfaces of edge AI. Section 3 describes the principles of prototype-based defense mechanisms and their theoretical foundations. Section 4 discusses architectural integration strategies for embedding prototype checks into edge inference pipelines. Section 5 analyzes the trade-offs among security, accuracy, latency, and resource consumption that arise from adopting prototype-guided defenses. Section 6 extends the discussion to governance, sustainability, fairness, and policy implications, drawing on case illustrations from multiple domains. Finally, Section 7 concludes with a summary of contributions and recommendations for future work.

2. Background and Threat Model

Distributed inference in edge AI typically follows one of two architectural paradigms: federated learning or split inference. In federated learning, each edge device trains a local model on its own data and periodically sends gradient or weight updates to a central server, which aggregates them to produce a global model [1]. Split inference, by contrast, partitions the model such that a shallow feature extractor runs on the device and the remaining layers execute on a more powerful edge server or cloud node [3]. Both architectures share a common

vulnerability: a malicious participant can manipulate the local update or the transmitted feature vector to poison the collaborative model. For instance, an attacker controlling a compromised device may craft a gradient that preserves the model's accuracy on normal inputs but introduces a backdoor that misclassifies inputs containing a specific trigger pattern [2].

The resource constraints of edge devices exacerbate the challenge of detecting such attacks. Devices typically possess limited RAM—often in the range of a few hundred megabytes—and modest CPU or microcontroller units that cannot support heavy cryptographic operations or complex statistical tests on high-dimensional parameter vectors [4]. Communication bandwidth is also scarce, especially in wireless networks with variable quality, making frequent exchange of full model updates or auxiliary verification data impractical. Furthermore, energy consumption must be minimized to preserve battery life, so any defense mechanism that requires additional computation or transmission cycles must be extremely efficient.

Existing defense strategies for model poisoning in federated learning, such as median-based or trimmed-mean aggregation, Byzantine-resilient consensus, and anomaly detection using distance metrics, have been shown to be effective in cloud settings [6]. However, these approaches often require the server to receive updates from all participants in each round, compute pairwise distances or order statistics, and then filter outliers—operations whose complexity grows quadratically with the number of clients. In edge networks where tens or hundreds of heterogeneous devices may join intermittently, such overhead becomes unaffordable. Similarly, cryptographic defenses like secure aggregation and zero-knowledge proofs impose latency and bandwidth penalties that degrade real-time inference performance [7].

The threat model considered here assumes a Byzantine adversary that controls a subset of edge devices, each of which can arbitrarily deviate from the prescribed protocol. The attacker's goal is to cause targeted misclassifications during inference while maintaining overall accuracy within an acceptable range to evade basic statistical filters. In split inference scenarios, the adversary may compromise the feature extractor on the device, sending corrupted intermediate representations to the server, or may compromise the server-side model, inserting backdoors into the classification head. Prototype-guided defense aims to detect such deviations without relying on a centralized, resource-heavy verification step. Instead, it leverages the fact that class prototypes—which can be computed during an initial calibration phase or updated incrementally—provide a stable reference distribution. Any parameter update or feature vector that falls far from its corresponding prototype in a chosen metric space signals potential poisoning and can be flagged for further inspection or rejection.

3. Prototype-Based Defense Mechanisms

Prototypes are representative examples or centroids that capture the essential characteristics of each class in a learned feature space [5]. In prototype learning, the model is trained to minimize the distance between input embeddings and the prototypes of their true class while maximizing distances to other class prototypes. This approach not only yields interpretable models—where each prediction can be explained by referencing the nearest prototype—but also creates an inherent structure that can be used for anomaly detection. The key insight for defense is that model poisoning attacks introduce perturbations that, while possibly preserving classification accuracy on benign inputs, tend to distort the alignment between local updates and the underlying prototype distribution.

The defense mechanism operates in two phases: initialization and detection. During initialization, a set of prototypes is computed for each class, either from a clean public dataset or from the initial global model before any compromised updates have been introduced. These prototypes can be the mean embedding of a small sample of correctly labeled examples, or they can be learned as part of the model architecture itself [8]. On each edge device, the prototype vectors are stored in a compact table—typically with dimensionality equal to the feature embedding size, which is usually several hundred dimensions—requiring only kilobytes of memory. In the detection phase, whenever a device transmits a local update (e.g., gradient or feature map), the receiving node (server or aggregator) checks the consistency of that update with the stored prototypes. For example, the cosine similarity between the update vector and the prototype of the predicted class can be computed; if the similarity falls below a threshold, the update is considered suspicious and either discarded or subjected to additional verification.

This approach is particularly well-suited for split inference, where the device sends a feature vector to the server. The server can compute the distance between that vector and the prototype of the class that the server's classifier predicts. A large deviation indicates that either the feature extractor is compromised or the input itself is out-of-distribution. In federated learning, prototypes can be used to assess the quality of gradients. Because benign gradients from a well-trained model tend to align with the direction that reduces prototype distances for the correct classes, malicious gradients that push in an orthogonal or opposite direction can be identified [9].

The theoretical grounding for prototype-guided defense lies in the notion of Lipschitz continuity and the stability of prototype representations. If the feature extractor is sufficiently smooth, small perturbations in the input space translate to bounded changes in the embedding space. Therefore, a sudden large change in the embedding—especially one that moves toward a different class prototype—is indicative of an attack. Moreover, prototypes can be updated adaptively to accommodate non-stationary data distributions, though such updates must be performed carefully to avoid introducing new vulnerabilities. A recent work has demonstrated that prototype consistency can be effectively used as a backdoor defense in vertical split learning, where the client and server hold different feature partitions [7]. This method, termed ProtoGuard-SL, enforces that the client's intermediate representations produce similar prototype distances under clean and backdoored conditions, thereby detecting trigger-based manipulations without requiring access to the original training data.

4. Architectural Integration in Edge AI

Embedding prototype-guided defense into existing edge AI architectures requires careful design to maintain the lightweight nature of the system. The integration should occur at two levels: within the local inference path on each device, and within the aggregation or coordination point (such as the edge server). On the device side, the prototype vectors can be stored in a read-only memory region that is updated only during scheduled calibration rounds. During inference, the device computes its feature embedding and, before transmitting, may optionally compute the prototype distance as a self-check. However, to minimize on-device computation, this self-check can be omitted, leaving the detection responsibility to the server. The server, being more powerful, can evaluate the prototype consistency for each incoming update or feature vector in constant time relative to the number of prototypes.

A critical architectural decision is how to set the detection threshold. A static threshold may fail under concept drift or natural distribution shifts, causing false positives that degrade

system availability. A dynamic threshold that adapts to recent observations can be implemented using a sliding window of past accepted updates; the mean and standard deviation of prototype distances are maintained, and an update is flagged if its distance exceeds a multiple of the standard deviation from the mean [10]. This approach preserves the lightweight memory footprint—only two scalar values per class need to be stored—while accounting for non-adversarial variability.

In federated edge networks, the aggregation server can combine prototype-based filtering with traditional robust aggregation methods. For instance, after discarding updates that fail the prototype consistency test, the remaining updates can be averaged using a trimmed mean to further guard against coordinated attacks. The computational overhead of this combined scheme is linear in the number of updates, as prototype checks require only a dot product and threshold comparison per update. Empirical studies in resource-constrained environments have shown that such hybrid approaches reduce the false positive rate while maintaining high detection sensitivity [11].

One must also consider the case where the prototypes themselves are corrupted—an attacker who compromises the calibration phase could inject malicious prototypes that accept poisoned updates. To mitigate this risk, prototypes should be computed from a trusted source, such as a secure enclave or a public verification dataset endorsed by a regulatory body. Alternatively, a decentralized consensus on prototypes among multiple edge servers can be reached using a lightweight Byzantine agreement protocol [12]. The cost of such consensus must be weighed against the security benefits; for low-stakes applications, a single trusted calibration may suffice.

5. Trade-offs and System Implications

Adopting prototype-guided defense introduces a set of trade-offs that system architects must navigate. The most immediate trade-off is between detection sensitivity and false positive rate. A low threshold may catch subtle attacks but will also reject benign updates caused by natural data variability, reducing the effective training data volume and potentially slowing convergence or degrading final accuracy [13]. A high threshold, conversely, may allow stealthy attacks to go undetected. Tuning this threshold requires domain-specific knowledge of the expected variance of embeddings, which can be estimated from a held-out validation set. In practice, a two-stage approach is often employed: a coarse filter with a loose threshold that eliminates straightforward outliers, followed by a more expensive inspection for the remaining suspicious cases.

Another critical trade-off involves memory and energy consumption. Although storing prototypes and computing distances is far cheaper than full model retraining or cryptographic verification, it still imposes a non-negligible overhead on ultra-low-power microcontrollers. For a model with a 256-dimensional embedding and ten classes, the prototype table occupies approximately 10 kilobytes. The distance computation for a single update requires 256 multiply-accumulate operations, which on a typical ARM Cortex-M4 processor consumes about 0.2 microjoules per embedding [14]. While small, this cost accumulates over millions of inferences in a battery-powered sensor node. Designers may choose to reduce the prototype dimension via principal component analysis or to compute distances only periodically rather than for every inference.

Latency is another dimension of concern. In real-time applications such as autonomous driving or industrial control, every millisecond of added delay matters. The prototype check

adds a small constant overhead—on the order of microseconds for modern edge CPUs—which is generally acceptable. However, if the threshold adaptation mechanism requires updating online statistics, the memory writes and potential synchronization overhead can increase latency. To mitigate this, statistics can be updated in a lazy manner or offloaded to a dedicated co-processor [15].

From a system reliability perspective, the deployment of prototype-guided defense must account for the possibility of false negatives—attacks that successfully bypass the filter. An attacker that can gradually drift the prototype distribution by injecting small, consistent perturbations over many rounds may remain undetected. This is analogous to the problem of slow poisoning in federated learning, where the adversary slowly bends the global model toward a malicious objective [16]. To counter this, the prototype update mechanism itself must be secured: prototypes should be updated only when a consensus of devices agrees that the distribution shift is benign, or they should be frozen after initial calibration.

6. Deployment, Governance, and Policy Implications

The practical deployment of prototype-guided defenses in edge AI raises broader socio-technical questions related to governance, sustainability, fairness, and accountability. One immediate governance challenge is the need for standardized testing and certification of prototype-based systems. Regulatory bodies, such as the Federal Communications Commission or the European Union Agency for Cybersecurity, may require that edge AI devices demonstrate resilience against model poisoning before being deployed in critical infrastructure [17]. A prototype consistency score could serve as a measurable security metric, similar to how differential privacy budgets are reported. However, there is no consensus yet on what constitutes an acceptable false positive rate or how to validate the security of prototype updates over the device lifecycle.

Sustainability concerns are also pertinent. The computational overhead of any defense mechanism increases the energy consumption and material footprint of edge devices. In massive IoT deployments comprising millions of sensors, even microjoule-level increments aggregate to significant environmental costs [18]. Prototype-guided defense, being computationally light, is more sustainable than alternatives like homomorphic encryption or secure multi-party computation, which can increase energy consumption by orders of magnitude. However, the production of the prototypes themselves—often requiring a small labeled dataset—may necessitate human annotation, which carries its own social and economic costs.

Fairness implications emerge from the potential for prototypes to encode biases present in the calibration data. If the calibration dataset does not adequately represent all demographic groups or operating conditions, the prototypes may disproportionately flag updates from underrepresented classes as anomalous, leading to systematic exclusion and degraded performance for those groups [19]. This is particularly concerning in applications like facial recognition or medical diagnosis, where equitable treatment is ethically mandated. To address this, the calibration process must be designed with diversity in mind, and the prototype distances should be normalized across classes to avoid threshold biases. Auditing mechanisms, such as fairness impact assessments, should be integrated into the deployment pipeline.

Policy frameworks must also consider the regulatory treatment of prototype updates. If prototypes are considered part of the model, any change to them could be subject to regulatory review, particularly in domains like healthcare or finance. The dynamic adaptation

of prototypes, while beneficial for handling concept drift, could be seen as model modification that requires re-certification [20]. A balanced policy might allow prototypes to be updated within a bounded region (e.g., the epsilon ball around the initial prototype) without triggering full re-evaluation, provided the update is verifiable via a tamper-proof log.

Case illustrations across domains underscore these points. In autonomous vehicle platooning, where edge AI nodes process sensor data for cooperative perception, prototype-guided defense can detect compromised nodes that attempt to inject false obstacle positions [21]. The low latency of prototype checks is essential for maintaining string stability. In smart healthcare, wearable devices that monitor vital signs and perform on-device diagnosis benefit from the privacy-preserving nature of prototype comparison—no raw data leaves the device, only the prototype distance. However, the risk of mis-calibration on individual patient data due to population-level prototypes must be mitigated through personalized fine-tuning [22]. In industrial IoT, predictive maintenance systems using split inference on programmable logic controllers can reject corrupted feature vectors from tampered sensors, but the harsh environmental conditions may cause natural embedding drift that requires careful threshold tuning.

7. Conclusion

Prototype-guided defense presents a viable and resource-efficient strategy for securing distributed inference in edge AI against model poisoning attacks. By leveraging class prototypes as lightweight, interpretable reference anchors, the system can detect anomalous updates without the computational and communication burdens of traditional defenses. This paper has provided a comprehensive analysis of the threat model, the underlying mechanism, architectural integration strategies, trade-offs, and broader socio-technical implications. The approach aligns well with the constraints of edge devices while offering a tunable security guarantee that can be adapted to application-specific requirements. Nonetheless, challenges remain, including the need for secure prototype initialization, robustness against slow poisoning, and fairness-aware calibration. Future research should explore adaptive prototypes that balance stability with plasticity, cross-device collaboration for prototype verification, and formal verification of security properties under resource bounds. As edge AI continues its penetration into critical infrastructure, defenses that are both effective and frugal will become indispensable; prototype-guided methods stand as a promising candidate in that endeavor.

References

1. McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, 54, 1273–1282.

2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., & Shmatikov, V. (2020). How to backdoor federated learning. Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics, 108, 2938–2948.

3. Vepakomma, P., Swedish, T., Raskar, R., Gupta, O., & Dubey, A. (2018). No need to know each other — Secure and communication-efficient split learning. arXiv preprint arXiv:1810.09115.

4. Blanchard, P., El Mhamdi, E. M., Guerraoui, R., & Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. Advances in Neural Information Processing Systems, 30, 119–129.

5. Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. Advances in Neural Information Processing Systems, 30, 4077–4087.

6. Yin, D., Chen, Y., Kannan, R., & Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. Proceedings of the 35th International Conference on Machine Learning, 80, 5650–5659.

7. Shui, Y., Jin, R., Dou, Z., & Gao, Z. (2026). ProtoGuard-SL: Prototype Consistency Based Backdoor Defense for Vertical Split Learning. arXiv preprint arXiv:2604.03595.

8. Li, W., Xu, Z., & Jiang, L. (2019). Attentive prototypes for image recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(8), 2032–2045.

9. Cao, D., & Wang, Y. (2021). Gradient-based prototype learning for backdoor detection in neural networks. Proceedings of the AAAI Conference on Artificial Intelligence, 35(8), 6856–6864.

10. Xie, C., Koyejo, O., & Gupta, I. (2020). Fall of empires: Breaking Byzantine-robust training by attacking the aggregation rule. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 11741–11750.

11. Chen, L., Zheng, Z., & Luo, X. (2022). Lightweight defensive distillation for edge AI. ACM Transactions on Embedded Computing Systems, 21(6), 1–22.

12. Guerraoui, R., & Huc, F. (2019). Asynchronous Byzantine consensus with trusted components. Proceedings of the 39th IEEE International Conference on Distributed Computing Systems, 405–415.

13. Baruch, G., Baruch, M., & Goldberg, Y. (2019). A little is enough: Circumventing defenses for distributed learning. Advances in Neural Information Processing Systems, 32, 8635–8645.

14. Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., & Kawsar, F. (2017). DeepX: A software accelerator for low-power deep learning inference on mobile devices. Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems, 1–14.

15. Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE, 105(12), 2295–2329.

16. Fang, C., Cao, Y., & Peng, W. (2020). Local and non-local defenses against model poisoning in federated learning. IEEE Transactions on Dependable and Secure Computing, 19(3), 1907–1920.

17. European Union Agency for Cybersecurity. (2021). Cybersecurity challenges in the uptake of artificial intelligence in the EU. ENISA Report.

18. Jones, N. (2018). How to stop data centres from gobbling up the world's electricity. Nature, 561(7722), 163–166.

19. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. ACM Computing Surveys, 54(6), 1–35.

20. U.S. Food and Drug Administration. (2021). Artificial intelligence and machine learning in software as a medical device. FDA Guidance.

21. Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3354–3361.

22. Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., ... & Cardoso, M. J. (2020). The future of digital health with federated learning. NPJ Digital Medicine, 3(1), 1–7.