A Taxonomy of Attacks and Defenses in Split Learning

Aqsa Shabbir[a,1], Halil İbrahim Kanpak[b,1], Alptekin Küpçü[b,*], Sinem Sav[a,*]

[a]*Bilkent University, Ankara, Türkiye*
[b]*Koç University, İstanbul, Türkiye*

Abstract

Split Learning (SL) has emerged as a promising paradigm for distributed deep learning, allowing resource-constrained clients to offload portions of their model computation to servers while maintaining collaborative learning. However, recent research has demonstrated that SL remains vulnerable to a range of privacy and security threats, including information leakage, model inversion, and adversarial attacks. While various defense mechanisms have been proposed, a systematic understanding of the attack landscape and corresponding countermeasures is still lacking. In this study, we present a comprehensive taxonomy of attacks and defenses in SL, categorizing them along three key dimensions: employed strategies, constraints, and effectiveness. Furthermore, we identify key open challenges and research gaps in SL based on our systematization, highlighting potential future directions.

Keywords: split learning, collaborative learning, distributed machine learning, privacy-preserving computation, data privacy

1. Introduction

The increasing adoption of Machine Learning as a Service (MLaaS) has revolutionized AI-driven applications across domains such as healthcare, finance, and IoT. However, ML outsourcing raises privacy concerns, as sensitive user data is often processed on external or untrusted servers [72]. To address this challenge, privacy-preserving machine learning (PPML) techniques have been developed, enabling collaborative learning while ensuring data confidentiality.

Among various collaborative methods, Split Learning (SL) emerged as a promising framework that partitions deep learning models between clients and servers [25]. SL enables clients to process several layers of a neural network locally while delegating the remaining layers to a server. This design reduces the computational burden on resource-constrained client devices while keeping raw data in the local premises of the client [81]. Over time, SL has evolved into different architectures, including Vanilla Split Learning (VanSL) [25], U-shaped Split Learning (USL) [25], No-Label Split Learning (NLSL) [45], Hybrid Split Learning (Hybrid-SL) [21], and Multi-hop Split Learning (MHSL) [81], each optimizing trade-offs between performance, scalability, and privacy.

Despite its architectural advantages for resource-constrained clients and inherent data locality [25, 81], SL is vulnerable to several security and privacy risks. Data reconstruction attacks exploit smashed data and gradients to infer private inputs, while label inference attacks analyze gradient updates to recover class labels [59, 16]. Property inference attacks extract sensitive attributes without full data reconstruction [52], and model inversion [10] aims to recover input samples from model activations. Additionally, backdoor and poisoning attacks [4, 90] manipulate the training pipeline to implant adversarial behavior. These vulnerabilities expose critical gaps in the privacy guarantees of SL, necessitating robust defense mechanisms to mitigate potential threats. However, before effective defenses can be designed and evaluated, a systematic understanding of the *attack landscape* itself is required. We build this foundation by addressing these core research questions regarding *attacks* against SL:

RQ1: How can adversarial objectives targeting SL be systematically taxonomized? Which architectural vulnerabilities in SL implementations serve as primary attack vectors?

RQ2: What are the fundamental characteristics and underlying principles of different SL attack mechanisms (e.g., feature-space hijacking, model inversion, gradient-based inference, embedding poisoning) that dictate their generalizability across SL variants and data domains, their potential for stealth, and their ability to bypass defenses?

RQ3: Under what operational constraints and adversarial assumptions (regarding knowledge, access, capabilities, and collusion scenarios) do attacks against SL demonstrate measurable effectiveness? How can their quantifiable impact on privacy and model integrity be systematically evaluated?

To counter these threats, researchers have explored various privacy-preserving techniques as defense strategies, including DP to perturb gradients [88], homomorphic encryption (HE) for encrypted computation to prevent unauthorized data access [61], and communication compression to reduce information leakage [62]. While each method strengthens SL security, it also introduces trade-offs in computational overhead, scalability, and model performance. Therefore, a clear understanding of the defense landscape, its capabilities, limitations, and evaluation is equally critical. Consequently, this paper also addresses these research questions concerning *defenses* in SL:

*Corresponding author.
E-mail addresses: akupcu@ku.edu.tr (A. Küpçü), sinem.sav@cs.bilkent.edu.tr (S. Sav).

[1]Equal contribution.

RQ4: What is the taxonomy of defense techniques against predefined attack scenarios/ adversarial objectives tailored for SL? Wat are the common techniques and tools being employed?

RQ5: Which defenses are suitable for a specific task? What are the constraints and conditions implied by these defenses? Can these techniques be optimized? Are these techniques suitable for real-world applications?

RQ6: How do defense techniques aim for success among different scenarios in terms of confidentiality or integrity of training? What constitutes a successful defense mechanism?

This paper provides a comprehensive SL analysis by systematically categorizing attack vectors and defense strategies based on these research questions. We propose taxonomies for attacks and defenses, evaluating their methodology, constraints, and effectiveness. We highlight key trade-offs, our observations, and open challenges derived from this systematization, paving the way for future advancements in SL privacy and robustness.

2. Review Scope and Methodology

We systematically categorize SL architectures, identify key attack surfaces —including data reconstruction, feature leakage, and adversarial interference— and evaluate existing defense mechanisms such as differential privacy, homomorphic encryption, and adversarial training. By consolidating insights from prior research, we provide a structured framework for assessing privacy risks, defense effectiveness, and security trade-offs, laying the foundation for future advancements in PPML.

Search Methodology. To ensure a comprehensive and representative collection of research on SL, we systematically gathered papers from major academic databases, including Google Scholar, IEEE Xplore, ACM Digital Library, arXiv, and SpringerLink. The search was conducted in two stages: an automated search using predefined keyword queries, followed by a snowballing approach to refine and expand the selection of relevant studies. We used keyword combinations to capture diverse SL research, including privacy, attack surfaces, and defenses. Example keywords included a combination of "*split learning*", "*privacy-preserving*", "*split neural networks*", "*privacy attacks on split learning*", "*homomorphic encryption based split Learning*", "*differential privacy based split learning*", and "*inference attacks in split learning*". These queries targeted research addressing both theoretical and practical aspects of SL, ensuring coverage of multiple perspectives on its security vulnerabilities and mitigation strategies. Then, we employed a snowballing approach, where we examined the references of key papers to identify prior work. Expanding this approach in both directions, we also examined papers that have cited the ones we identified, allowing us to trace the evolution of knowledge in the field.

After identifying works across these different feature aspects, we aligned and classified them to develop our overall taxonomy for SL systematization. During the filtering process, each identified paper was first assessed based on its abstract, followed by a thorough review of its methodology and experimental evaluation if the abstract indicated the use of a relevant technique. Papers that did not explicitly reference SL, such as those focused on general distributed machine learning, were excluded. We then classified each paper as comprehensively as possible, considering multiple aspects such as the defense mechanisms it proposes in case of an attack paper and the attacks it addresses in case of a defense paper, the assumed threat model, and the server-client setup.

Systematization Methodology. For our systematization, we begin by reviewing SL architectures to establish a foundational understanding of the landscape. We then categorized the collected papers into two primary groups: attack papers and defense papers. Within each group, we employ a structured classification approach based on three key dimensions:

1. Strategies: The method or strategies used to conduct attacks or implement defenses.
2. Constraints: The assumptions, limitations, and requirements under which the attacks or defenses operate.
3. Effectiveness: The impact and success rate of attacks, as well as the robustness and trade-offs of defense mechanisms.

Finally, we present a timeline in Figure 1 that compiles our surveyed literature on both attacks and defenses. It illustrates the chronological progression and dominant strategies.

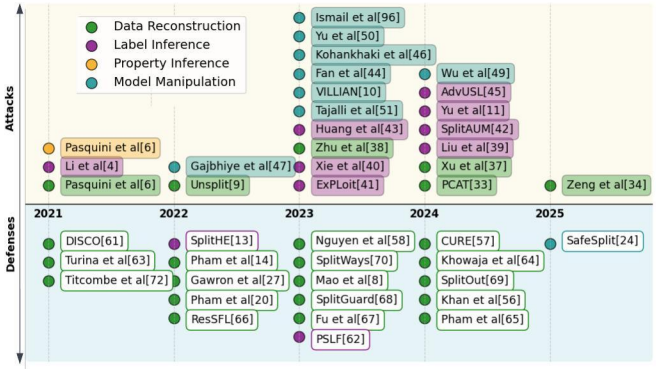

Figure 1: SL attack and defense timeline (2021–2025), with attacks on the top half and defenses on the bottom half. Attack and defense types are categorized into four groups, as shown in the legend. Attacks are represented with a colored background, while the corresponding defenses share the same color but are displayed without a background.

3. Split Learning (SL)

We provide an overview of split learning (SL), its variants, and key concepts. We provide the frequently used notations and abbreviations in Table 1.

3.1. Split Learning and Its Variants

SL [25] is a distributed machine learning paradigm where the computation of a deep learning model is split between clients and servers. It has evolved into various forms to address the diverse needs of collaborative machine learning. The key variants include Vanilla SL [25], U-shaped SL [25], no-label SL [45],

Table 1: Key Notations and Abbreviations

Notation/Abbreviation	Description
f_c	Client-side network model part
f_s	Server-side network model part
x	Raw input data (client-side)
y	True label (often client-side)
z_c	Smashed data
∇_{z_c}	Gradient sent from server to client
$\hat{y}$	Model's final output/prediction
$\tilde{x}$	Reconstructed input data
$\tilde{y}$	Inferred label
VanSL	Vanilla Split Learning
USL	U-Shaped Split Learning
SFL	Split Federated Learning
HSL	Horizontal Split Learning
VSL	Vertical Split Learning
SCSL	Single-Client Split Learning
MCSL	Multi-Client Split Learning
SSSL	Single-Server Split Learning
MSSL	Multi-Server Split Learning
SHS	Semi-Honest Server
MS	Malicious Server
MC	Malicious Client
SHC	Semi-Honest Client
FL	Federated Learning
HE	Homomorphic Encryption
DP	Differential Privacy

Hybrid SL [76], and Multi-hop SL [81]. Additionally, the effectiveness of SL can be influenced by the choice of data partitioning schemes, namely horizontal partitioning and vertical partitioning, which dictate how data is distributed among participants. We detail these variants below and illustrate the most commonly used ones in Figure 2.

Vanilla Split Learning (VanSL): VanSL is the simplest form of SL, where the client processes the *initial layers* (f_c), and the server processes the *remaining layers* (f_s). The client uses its private data x to compute the *intermediate activations* $z_c = f_c(x)$, which are transmitted to the server. The server completes the *forward pass* by calculating the output $\hat{y} = f_s(z_c)$. During backpropagation (f'), the server computes the gradient of the loss function L with respect to z_c, denoted as $\nabla_{z_c} = \frac{\partial L}{\partial z_c}$, and sends it back to the client. The client then updates the parameters of f_c using its local gradients. This collaborative process ensures the client retains x.

U-shaped Split Learning (USL): USL divides the model into three segments: (i) *initial layers* f_c processed by the client, (ii) *middle layers* f_s handled by the server, and (iii) *remaining layers* f_{c_r} completed by the client. The client begins by processing x through f_c to compute z_c and sends z_c to the server for further computation. The server processes z_c through f_s to compute z_s and returns z_s to the client. The client completes the forward pass by applying f_{c_r} to z_s, resulting in the final output $\hat{y}$. The backpropagation process follows the same sequence with a similar approach, in reverse. This way, USL ensures that x and $\hat{y}$ remain at the client while enabling end-to-end training.

Hybrid Split Learning: Hybrid SL integrates SL with other collaborative learning paradigms, such as federated learning (FL) to enhance privacy, scalability, and efficiency. For instance, combining SL with FL, Split Federated Learning (SFL) allows clients to train parts of the model collaboratively while leveraging federated learning aggregation techniques. Similarly, incorporating HE ensures that smashed data exchanged between clients and servers remains encrypted, further strengthening privacy. It enables solutions for scenarios involving heterogeneous clients and large-scale data distributions. However, integrating additional mechanisms often introduces computational and communication overhead, making efficient design and implementation critical for practical use.

We present less commonly adopted SL variants, i.e., **multi-hop** and **no-label SL**, in Supplementary Material A. Finally, unlike Federated Learning (FL) [94], where the entire model is trained locally on each client and the gradients are aggregated by a server, SL partitions the model, delegating the more computationally intensive layers to the server. This makes SL [81] ideal for resource-constrained clients, who wish to outsource most of the computationally-intensive training procedure.

3.2. Data Partitioning Schemes

Data partitioning in SL determines how data is distributed among clients, impacting both the training process and privacy assurances. In **Horizontal Split Learning (HSL)** [47, 81], clients handle data with identical feature spaces but different samples; for instance, multiple hospitals with similar patient data but distinct patient groups can collaboratively train a model by sharing intermediate activations without exposing raw data. Conversely, **Vertical Split Learning (VSL)** [81, 34] is applicable when clients possess complementary features for the samples; each client processes its unique features, and a server combines these embeddings to facilitate joint learning, such as a bank and an e-commerce platform sharing distinct user attributes to jointly develop a credit scoring model.

3.3. Client-Server Participation Models

Single Client Split Learning (SCSL) In SCSL [80], training involves one client and a server, making it ideal for a single organization, such as a hospital or research institute, that seeks to preserve privacy while offloading the heavy computation to a server. Although it simplifies communication and coordination, clients still need sufficient resources to process their local model portions.

Multi Client Split Learning (MCSL) MCSL [65] keeps the forward and backward propagation split between multiple clients and a server. Clients only send smashed data to the server, and the server processes the deeper layers of the model and updates all clients without requiring them to maintain a full model locally.

Single Server Split Learning (SSSL) In SSSL [81], all clients interact with a single server. Clients send intermediate activations to the server, which processes the remaining model layers

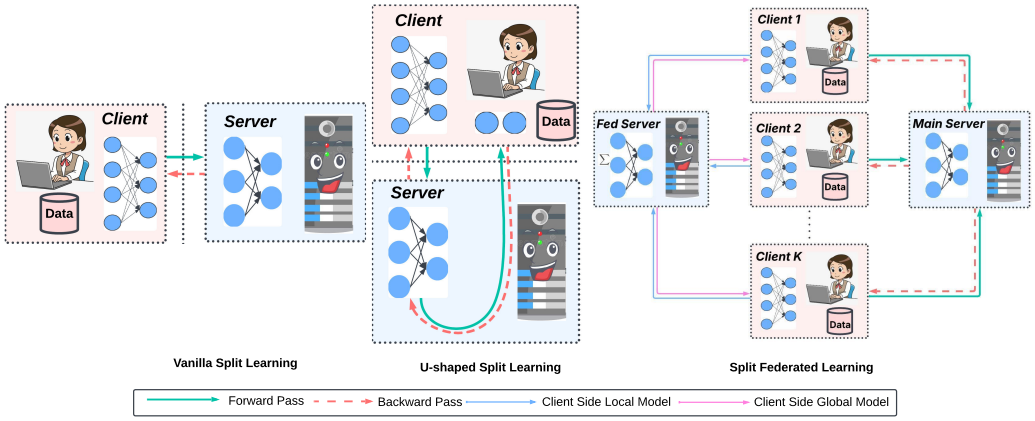

Figure 2: Overview of three key SL variants: Vanilla Split Learning, U-shaped Split Learning, and Split Federated Learning. The figure highlights their architectural distinctions, client-server roles, and the flow of forward and backward passes in training.

and computes gradients. The server then returns these gradients to clients. Unlike MCSL, where multiple clients interact with multiple servers, SSSL uses a single-server architecture with all clients relying on one server for training. This setup is advantageous for organizations with centralized infrastructure capable of handling high computational demands. It simplifies system architecture and reduces coordination overhead but may introduce scalability limitations as the number of clients increases.

Multi-Server Split Learning (MSSL) MSSL [65] divides the model computation across multiple servers, allowing better scalability and resource utilization. Each client processes the initial model layers locally and sends the smashed data to one or more servers. The servers handle subsequent computations, often partitioning the model layers or tasks among themselves, and collaboratively complete forward and backward passes. MSSL is ideal for large-scale deployments where a single server cannot handle the computation or communication demands of multiple clients. It enhances system scalability and fault tolerance but requires efficient communication and synchronization between servers.

3.4. Positioning SL within the PPML Landscape

While SL exposes intermediate representations to the server, leading to attacks such as model inversion, label inference, and feature space hijacking, other PPML methods, such as Federated Learning (FL) and Secure Aggregation, are not immune to similar threats. For instance, in FL, gradients shared by clients can be exploited to reconstruct raw inputs, even without access to the model architecture or full training data [95]. Similarly, Secure Aggregation protocols may be circumvented when adversaries collude or exploit partial gradient information to infer sensitive data [28, 54]. These observations suggest that the underlying vulnerability (exposure of sensitive gradients or activations) is a broader issue in collaborative learning, not one unique to SL. Nonetheless, its advantages and disadvantages must be weighed carefully.

One of SL's key strengths is offloading the client's computational burden to the server. In standard setups such as Vanilla SL or U-shaped SL, the client only processes a few initial layers of the model and never needs to store or train the full network [25, 81]. This makes SL particularly suitable for resource-constrained devices such as smartphones, wearables, or embedded IoT systems in domains like digital health, where data confidentiality is crucial and local compute capacity is limited.

However, SL also has its limitations: It lacks robust mechanisms for verifying the integrity of the training process, leaving it vulnerable to model manipulation, data poisoning, and backdoor attacks, especially in multi-client settings where colluding parties can exploit sequential training dynamics [4, 45, 69]. Moreover, SL's training is often serialized, particularly in multi-client architectures, leading to communication bottlenecks and slower convergence compared to methods that allow concurrent client updates [65].

Given these trade-offs, SL is most appropriate in scenarios where deep models are essential, client computational power is constrained, and data sensitivity is high. Examples include natural language processing tasks involving transformer-based models or clinical settings with privacy-sensitive patient records [70]. Poirot et al. [68] demonstrated the effectiveness of SL in healthcare across disparate datasets while preserving privacy. Wang et al. [84] explored Stitch-able SL for multi-UAV systems, showcasing its ability to handle device instability and model heterogeneity in complex tasks. Li et al. [46] applied Split Learning to BERT-based models, achieving communication efficiency and privacy preservation in decentralized text classification.

4. Attacks on Split Learning

SL vulnerabilities arise from shared intermediate activations, gradients, and model states, exposing potential attack surfaces. This section categorizes these threats across three dimensions: *attack strategies* (Section 4.1), *operational constraints* (Section 4.2), and *attack effectiveness* (Section 4.3). Finally, we provide a summary of the surveyed literature in Table 2, Supplementary Material E.

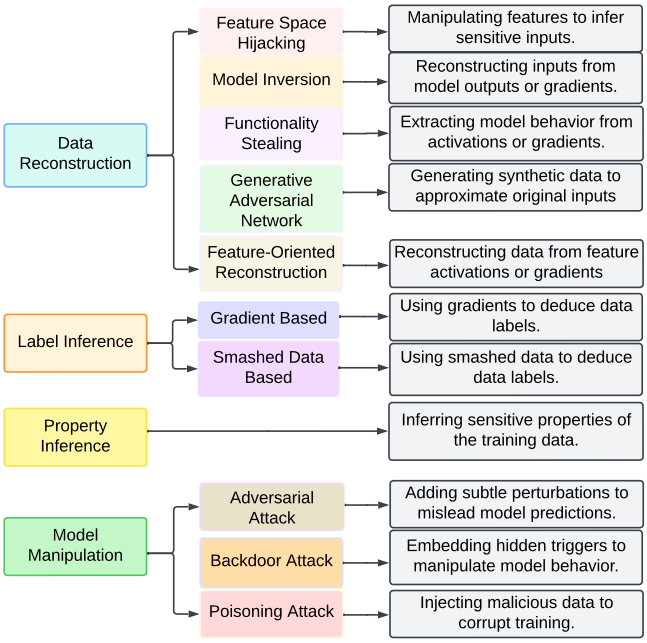

Figure 3: A taxonomy of attacks in SL categorizing attack vectors into: (1) Data Reconstruction, (2) Label Inference, (3) Property Inference, and (4) Model Manipulation Attacks.

4.1. Attack Strategies

The attacks exploit specific ML principles and methodologies. In this section, we categorize them into four main strategies: data reconstruction attacks (Section 4.1.1), label inference attacks (Section 4.1.2), property inference attacks (Section 4.1.3), and model manipulation attacks (Section 4.1.4). We further refine these categories into subcategories in Figure 3, which presents a structured taxonomy of attacks, systematically highlighting key attack vectors. Additionally, Figure 4 illustrates the distribution of adversarial attack models.

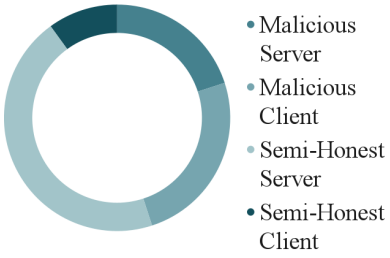

Figure 4: The distribution of adversarial models of surveyed attacks in SL.

4.1.1. Data Reconstruction Attacks

Data Reconstruction Attacks represent a significant privacy vulnerability in SL systems, wherein the adversary attempts to recover sensitive training data. These attacks leverage model characteristics, including gradients, confidence scores, and internal representations, to reconstruct private training inputs. The adversary optimizes a reconstruction function to approximate the original input x, formulated as:

$$\tilde{x} = \arg\min_{\tilde{x}^*} \mathcal{L}_{\text{rec}}(f_c(\tilde{x}^*), z_c) + \lambda R(\tilde{x}^*) \quad (1)$$

where $\tilde{x}$ is the reconstructed input, and $\tilde{x}^*$ is iteratively refined to align with x. The reconstruction loss $\mathcal{L}_{\text{rec}}$ minimizes the difference between the reconstructed and original feature representations. At the same time, regularization $R(\tilde{x}^*)$ imposes constraints such as Total Variation (TV) loss [71] or adversarial regularization [53]. The term λ controls the balance between accuracy and constraint enforcement. The primary methodologies are (i) Feature Space Hijacking Attack (FSHA), (ii) Model Inversion Attack, (iii) Functionality Stealing, (iv) Generative Adversarial Network (GAN), and (v) Feature Reconstruction. We further provide detailed explanations of each subtype of data reconstruction attack in Supplementary Material B.

4.1.1.1 Feature Space Hijacking Attack (FSHA). FSHA [59, 22] is a data reconstruction attack that exploits feature representations in SL settings. By leveraging adversarial learning, a technique where models are trained using adversarial examples, intentionally crafted inputs designed to mislead the model, FSHA enables the adversary to approximate private client data from the exchanged feature space. This attack employs a three-component system: a pilot network ($\tilde{f}_c$) that defines the target feature space, an inverse network ($\tilde{f}_c^{-1}$) that reconstructs inputs from features, and a discriminator (D) that guides the mapping learning through adversarial training.

4.1.1.2 Model Inversion:. Model Inversion is an attack technique where an adversary attempts to reconstruct private training data by exploiting a model's parameters, outputs, or gradients. As demonstrated by [27, 10], a malicious party can leverage the information received during the SL process to recover client data. This approach involves an optimization procedure where the adversary generates synthetic inputs that produce activations or gradients matching those observed during training when fed through the model. By iteratively minimizing the distance between these synthetic outputs and the observed ones, often using mean squared error (MSE) as the objective function, the adversary can approximate the original private data.

4.1.1.3 Functionality Stealing:. Functionality stealing [92] is an attack where an adversary replicates a model's behavior. By mimicking its functionality, the attacker approximates the original model and extracts sensitive information. The adversary can train a pseudo-client model ($\tilde{f}_c$) that functionally mimics f_c without knowing its structure. Once the functionality is stolen, the adversary can train a reverse mapping function to transform features back into x, effectively compromising client privacy across multiple clients without modification.

4.1.1.4 Generative Adversarial Network (GAN). GANs have emerged as powerful tools for data reconstruction attacks [91, 52] in SL environments. This approach involves training the generator G to produce synthetic inputs that yield feature representations or gradients matching those observed during training. The discriminator D helps refine these reconstructions by providing feedback on their realism. GAN-based attacks are concerning because they can operate effectively with limited information and can progressively improve reconstruction quality through the adversarial training process.

4.1.1.5 Feature Reconstruction. Feature reconstruction is an attack that aims to recover the original input data from intermediate feature representations exchanged during model training. By leveraging statistical measures [24, 49] and adversarial learning, attackers can approximate private data to exploit the representation preferences encoded in z_c [87, 96].

4.1.2. Label Inference Attacks

Label inference attacks exploit the correlation between model updates and labels to infer client information. By analyzing shared gradients or smashed data during training, these attacks leverage inherent patterns in the learning process to reconstruct labels. They are primarily categorized into gradient-based and smashed data-based label inferences.

4.1.2.1 Gradient-Based Label Inference:. These attacks exploit the correlation between gradient updates and label distributions to infer client information. We provide a detailed explanation of gradient-based label inference attacks in Supplementary Material C.1. Three methods dominate the literature for performing gradient-based label inference attacks: (i) similarity-based techniques [48, 45] that compare gradients directly by selecting the expected label (y_{exp}) and minimizing the difference between the observed gradient (∇_c) and expected gradient (∇_{exp}):

$$\tilde{y} = \arg\min_{y_{\text{exp}}} \|\nabla_c - \nabla_{exp}\|^2 \tag{2}$$

where $\|\cdot\|$ represents the similarity metric, such as Euclidean norm (ℓ_2-norm) or cosine similarity. (ii) loss function-based inference [10, 86] that iteratively refines predictions in searching for the most expected label y_{exp} by minimizing MSE or cross-entropy loss:

$$\tilde{y} = \arg\min_{y_{\text{exp}}} \mathcal{L}_{\text{adv}}(\nabla_c - \nabla_{exp}) \tag{3}$$

where ($\mathcal{L}_{\text{adv}}$) is the adversary loss function measuring the difference between ∇_c and ∇_{exp}, and (iii) surrogate model optimization [37, 93, 4], where an adversary iteratively refines the estimated label by minimizing the loss function using gradient-based optimization. Unlike direct gradient matching or similarity-based inference, this method leverages a surrogate model to approximate the relationship between x, ∇, and y, allowing for a structured reconstruction of private labels. At each iteration t, the adversary updates the estimated label $\tilde{y}(t)$ by performing a gradient step that minimizes the loss function:

$$\tilde{y}^{(t+1)} = \tilde{y}^{(t)} - \eta \frac{\partial \mathcal{L}\text{adv}(\nabla_c, \nabla exp(\tilde{y}^{(t)}))}{\partial \tilde{y}} \tag{4}$$

where η is the learning rate that controls the step size for updating the label estimate.

4.1.2.2 Smashed Data-Based Label Inference:. In smashed data-based label inference attacks, an adversary aims to reconstruct private labels by analyzing the structural and semantic properties of z received by the adversary in an SL setup. Unlike gradient-based attacks, these methods do not require gradient updates but instead rely on the inherent information encoded in z. The adversary infers labels by minimizing a predefined loss function ($\mathcal{L}_{\text{adv}}$), which measures the difference between the observed z and reference embeddings (z_{exp}). This general process is formulated as:

$$\tilde{y} = \arg\min_{y_{\text{exp}}} \mathcal{L}_{\text{adv}}(\mathcal{F}(z), \mathcal{F}(z_{\text{exp}})) \tag{5}$$

where, $\mathcal{F}$ represents the adversarial function. We detail the smash-based label inference attacks in Supplementary Material C.2. Researchers have explored three primary approaches for label inference leveraging smashed data in SL: (i) distance-based matching [48], which measures the similarity between observed and reference embeddings using metrics like Euclidean distance; (ii) clustering-based inference [48, 96], which groups similar embeddings to identify patterns in labels; and (iii) transfer learning-based inference [48, 30], which leverages pre-trained models or generators to extract label information from embeddings. These techniques demonstrate that even without gradient information adversaries can successfully infer private labels by exploiting the semantic properties preserved in the shared intermediate representations.

4.1.3. Property Inference Attacks

Property inference attacks extract sensitive attributes or statistical patterns from client data without requiring complete data reconstruction. Unlike reconstruction attacks, these approaches focus on inferring specific properties such as demographic information, class distributions, or other sensitive characteristics embedded within the private dataset. As demonstrated in [59, 52], property inference attacks in SL typically leverage the intermediate representations or model updates shared during the collaborative training process. The approach involves an adversary creating a specialized classifier (C) that maps the observed smashed data or gradients to property labels. By analyzing patterns in the information, an adversary can infer properties that were never intended to be shared. The attack can be formulated as:

$$A_{PIA} = \arg\min_{F} \sum_{x_i \in X} \mathcal{L}_{adv}(F(T(x_i)), l_i), \quad l_i \in 0, 1 \tag{6}$$

where (F) represents the adversarial inference model, (T) is the function that processes client data and produces the observable artifacts, and (l_i) indicates the property being inferred. These attacks are particularly concerning because they can reveal sensitive information while the participating entities believe they are only sharing task-relevant features.

4.1.4. Model Manipulation Attacks

This section explores attacks that compromise SL model integrity, categorizing them into: (i) adversarial attacks that perturb inputs to cause misclassification, and (ii) backdoor and poisoning attacks that manipulate training data to induce malicious behavior or degrade performance (see Supplementary Material D).

4.1.4.1 Adversarial Attacks.. Adversarial attacks manipulate intermediate feature representations by introducing perturbations that induce misclassification. These attacks can be categorized as non-targeted attacks [14] and targeted attacks [26]. We provide a summary of adversarial attacks and detail them in Supplementary Material D.1. The objective of a non-targeted attack is to maximize the difference between the perturbed representation and the clean feature representation z. This is achieved by often using cosine similarity or Euclidean distance:

$$\xi^* = \arg\max_{\|\xi\|_\infty \leq \epsilon} \mathcal{L}_{\text{attack}}(z, \xi + z) \tag{7}$$

where ξ^* is the optimal adversarial perturbation, ξ is the adversarial perturbation applied to z, and $\|\xi\|_\infty \leq \epsilon$ ensures that the perturbation remains within the allowed bound. However, in a targeted attack, the adversary modifies z such that they move toward a predefined target embedding z_t, forcing the model to produce a specific incorrect prediction. This is achieved by minimizing (rather than maximizing) the attack loss function in Equation 7. The reason is that in a non-targeted attack, the adversary aims to *maximize* the deviation of the perturbed representation from the original feature representation, making the output unpredictable and unreliable. In contrast, a targeted attack seeks to *minimize* the difference between the perturbed representation and a specific target representation z_t, effectively steering the model toward a controlled misclassifications.

4.1.4.2 Backdoor and Poisoning Attacks.. SL models are vulnerable to adversarial manipulations that exploit weaknesses in the training process, particularly through poisoning attacks and backdoor attacks. *Poisoning attacks* manipulate the training process by modifying the dataset to degrade model performance. Let x, y denote the original training data and labels, and $\mathcal{A}(x, y)$ be the adversarial poisoning function that generates a poisoned dataset x', y'. The model, parameterized by θ, updates its parameters to θ^* after training on poisoned data, leading to incorrect decision boundaries. The adversary's objective is to maximize classification errors, formulated as a minimization problem where the model, trained on poisoned data, unknowingly optimizes its loss function in a way that degrades generalization and increases misclassification:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x',y')\sim\mathcal{A}(x,y)} \mathcal{L}(f_\theta(x'), y') \tag{8}$$

where $f_\theta(x')$ is the model's output, and $\mathcal{L}(f_\theta(X'), Y')$ measures the classification loss on the poisoned dataset. The expected loss function $\mathbb{E}_{(x',y')\sim\mathcal{A}(x,y)}$ ensures the poisoning effect generalizes across training samples. *Backdoor attacks* implant hidden triggers into the training process, enabling adversaries to manipulate model predictions when the trigger is present while maintaining normal behavior otherwise. Let x, y denote the clean training data and labels, and let x_b be the backdoor-inserted inputs with the attacker-defined target labels y_t. The adversary optimizes a dual-objective function that ensures normal classification on clean samples while inducing misclassification on backdoor samples:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{clean}}} \mathcal{L}(f_\theta(x), y) + \lambda \mathbb{E}_{(x_b,y_t)\sim\mathcal{D}_{\text{b}}} \mathcal{L}(f_\theta(x_b), y_t)$$

where $\mathcal{D}_{\text{clean}}$ represents the clean dataset used for normal training, while $\mathcal{D}_{\text{b}}$ contains samples modified with a trigger. The adversary applies a backdoor function to x, producing x_b that, when processed by the model, leads to a specific misclassification. The attacker assigns target labels y_t to these backdoor-embedded samples, ensuring that normal inputs retain their correct classification while triggered inputs are classified incorrectly. λ is a regularization parameter that balances the model's ability to maintain the accuracy on clean inputs while embedding the backdoor functionality (see Supplementary Material D.2 for details of backdoor and poisoning attacks.)

a. Label Flipping Attacks: Label flipping attacks represent a fundamental poisoning strategy in SL, where adversaries deliberately alter class labels within the training data [43, 19, 32]. These attacks are executed by malicious clients who manipulate their local datasets before participating in the collaborative training process. The general approach involves systematically modifying the true labels of training samples to incorrect ones, either through targeted flipping (changing specific source classes to pre-selected target classes), untargeted flipping (random mislabeling across multiple classes), or distance-based optimization (selecting target classes that maximize classification errors based on feature proximity). The effectiveness of label flipping attacks increases with the poisoning rate. For instance, even 10% malicious clients reduced class recall by 12.58% on CIFAR-10 and up to 47.31% with 50% malicious clients [19]. Similar trends were observed by Ismail and Shukla [32], where untargeted and distance-based attacks caused greater accuracy degradation than targeted attacks in both MNIST and ECG datasets, confirming their higher impact across domains.

b. Embedding Poisoning Attacks: Embedding poisoning attacks target the smashed data, manipulating the embedding space rather than raw data or labels [4, 85]. The general approach involves introducing perturbations to the feature representations ($\tilde{z}_i$) before they are transmitted to the server. These perturbations are designed to be subtle enough to avoid detection while causing the model to learn unintended patterns or vulnerabilities. These attacks are effective because they directly compromise the information exchange that is fundamental to the collaborative learning process.

c. Client-Side Backdoor Attacks: Client-side backdoor attacks in SL involve malicious clients who inject hidden triggers into their local training data or model components. These attacks follow a pattern where the adversary modifies a subset of their training data to include specific trigger patterns associated with targeted misclassifications [89, 69]. The approach involves training the local model component to respond to these triggers while maintaining normal performance on clean data. This can be achieved through feature manipulation, label modification, or the use of auxiliary models to distinguish between clean and backdoored samples. The distributed nature of the system makes it difficult to detect malicious behavior, allowing backdoors to persist across multiple training rounds while maintaining model utility for the primary task.

d. Server-Side Backdoor Attacks: In server-side backdoor attacks, a malicious server compromises the integrity of the SL model. As investigated in [74, 89, 90], these attacks lever-

age the server's privileged position in the training process to implant backdoor functionalities without direct access to client data. The methodology involves manipulating the shared model components or gradient updates to create hidden vulnerabilities that can be exploited later. This can be achieved through surrogate model training, feature space manipulation, or strategic modification of model updates.

4.2. Attack Constraints

Attack constraints define the technical and operational prerequisites for adversaries to execute attacks in SL successfully.

Dataset Constraints. Dataset constraints influence the vulnerability landscape of SL systems. One fundamental constraint is the adversary's ability to access domain-similar datasets, which substantially enhances the feasibility of various attack vectors [59]. Studies have shown that an exact match between the adversary's dataset and the target data is not necessary; it is sufficient for the datasets to share similar distributional characteristics to mount effective attacks [87, 26, 96, 30]. Furthermore, access to publicly available datasets enables the development of more sophisticated threats. For instance, adversaries can train surrogate models to reconstruct private data [92, 91] or employ shadow models to conduct backdoor insertion and poisoning attacks [90, 89, 32].

Knowledge-Based Constraints. The assumption that the server possesses complete knowledge of the learning task significantly amplifies the effectiveness of adversarial strategies. Under this constraint, adversaries are able to design task-specific queries that exploit the model's learning objective, thereby increasing the risk of privacy breaches [59, 92]. For instance, in a setting where the learning task involves medical imaging, adversaries can synthesize patient-specific scans to infer sensitive attributes. Additional assumptions, such as the server and client sharing the same optimizer, further facilitate attack success by enabling more accurate reconstruction of training data or inference of model parameters [92]. Some works explore scenarios where the adversary operates with only partial information about the learning objective [14].

Architecture Exposure Constraints. Architecture exposure constraints play a crucial role in determining the vulnerability of SL systems to various attacks. The exposure to client-side architecture significantly enhances the efficacy of model manipulation [26, 30], inversion [10], and feature reconstruction attacks [96]. Further expanding on this constraint, more comprehensive threat models have been proposed, where adversaries are assumed to have full knowledge of both the client's subnetwork architecture and the dataset distribution [74]. This enhanced architectural exposure notably strengthens the potential for backdoor attacks, enabling adversaries to manipulate the model's internal structure and implement covert functionalities while evading detection.

Label and Classification Constraints. Knowledge of the number of discrete labels in the dataset allows adversaries to refine their predictions and narrow the search space for classification inference [10]. In some settings, adversaries are further assumed to employ binary classifiers to predict labels from observed model activations, a technique commonly leveraged in membership inference attacks to determine the presence of specific samples in the training set [45]. Additional assumptions, such as requiring only a single labeled sample per class or having access to all participant labels, further strengthen the adversary's capability to infer broader label information from model activations [48, 4].

4.3. Attack Effectiveness

Domain-Independent Attacks. Such attacks do not rely on specific datasets, architectures, parameter distribution, or optimization methods. Unlike traditional attacks that exploit domain-specific patterns, these attacks leverage fundamental weaknesses in the SL process, allowing them to generalize across multiple settings. For instance, [59, 92, 87, 48, 45, 37, 4, 10] attacks have been shown to reconstruct input data and inference labels across diverse learning environments, demonstrating that their effectiveness is not constrained to a particular application.

Stealth. In traditional security frameworks, attacks are often identified by monitoring anomalies in system behavior. However, many SL attacks [59, 92, 10, 87, 48, 90, 26, 96, 93, 69, 86, 19, 32] evade detection by ensuring that they do not introduce observable deviations in gradient updates or model performance. For instance, distance correlation minimization is leveraged in [59, 10, 87] to launch stealthy attacks, where gradient updates are subtly manipulated while preserving the overall model behavior.

Defeating Differential Privacy. Differential Privacy (DP) [9] is recognized as one of the most effective privacy-preserving methods in machine learning. It operates by introducing controlled random noise into training updates, thereby ensuring that the inclusion or exclusion of any individual data point does not significantly alter the model's output (detailed in Section 5.1.1). This mechanism provides theoretical guarantees that adversaries cannot precisely infer the presence of a specific data sample. However, recent studies [22, 92, 45, 48, 4, 93, 30] have demonstrated that DP alone is insufficient in defending against advanced SL attacks. The primary limitation of DP in the SL setting is that it focuses on protecting individual data contributions rather than preventing full-scale data reconstruction. While DP effectively limits the sensitivity of activations and gradient updates, adversaries can still exploit structural and statistical properties of gradients to recover private information at a more aggregated level. Moreover, adversaries employing distance correlation minimization techniques [59, 10, 87] have been shown to circumvent DP protections. By iteratively optimizing gradient representations, adversaries can reconstruct highly accurate approximations of the original input data, reducing the effectiveness of DP.

SL Variants. The effectiveness of attacks in SL is not confined to a single variant but extends across all architectures [92, 14, 96, 93], demonstrating a fundamental vulnerability. Since all SL variants inherently rely on exchanging intermediate activations or gradients between the clients and the servers, adversaries can exploit this communication to infer sensitive information, manipulate learning dynamics, or introduce adversarial perturbations. The structural modifications between VanSL, USL, and Hybrid SL do not provide sufficient protection, as the core vulnerability (gradient leakage and model influence) remains consistent across variants.

5. Defense For Split Learning

In Section 4, we explored how adversaries exploit SL vulnerabilities. In response, various defense strategies have emerged. This section systematically analyzes defense mechanisms based on *defense strategies* (Section 5.1), *operational constraints* (Section 5.2), and *effectiveness* (Section 5.3), with a summary in Table 2, Supplementary Material E.

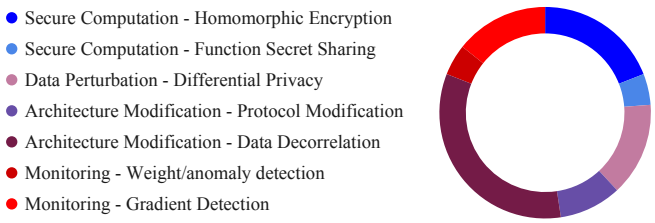

Figure 5: The distribution of defense mechanisms classified in the surveyed literature.

5.1. Defense Strategies

In this section, we present defense strategies, first categorizing them into protection and detection mechanisms, followed by their subcategories. The distribution and taxonomy of these strategies in the surveyed literature is illustrated in Figures 5, and 6, respectively.

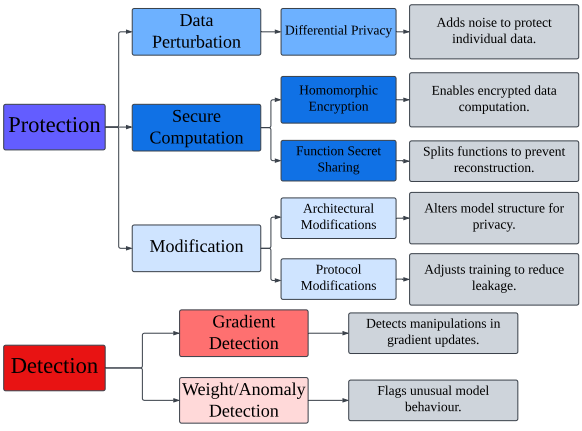

Figure 6: Taxonomy of Defense Strategies

5.1.1. Protection Mechanisms

Protection mechanisms aim to secure parties preemptively before attacks occur. These approaches generally conceal or limit the information an adversary can extract, often assuming a *semi-honest* threat model.

5.1.1.1 Data Perturbation:. In this approach, clients intentionally *modify* (e.g., add noise to) data or intermediate representations, aiming to obscure sensitive information while preserving sufficient utility for training. *Differential privacy* (DP) gained prominence due to its well-defined mathematical framework that achieves provable formal privacy guarantees. By calibrating noise according to (ε, δ)-DP, data contributors can formally bound the risk. The privacy guarantee is established through the definition in Equation (9), ensuring that even if an adversary observes these shared activations, they cannot confidently distinguish whether a specific data point was included in the dataset $\mathcal{X}$.

$$\Pr[\mathcal{M}(\mathcal{X}) \in S] \leq e^{\varepsilon} \Pr[\mathcal{M}(\mathcal{X}') \in S] + \delta. \tag{9}$$

where $\mathcal{M}$ is a randomized mechanism that takes a dataset $\mathcal{X}$ as input and $\mathcal{M}$ satisfies (ε, δ)-differential privacy for all neighboring datasets $\mathcal{X}$ and $\mathcal{X}'$ differing in at most one entry. Here, ε defines the privacy budget and δ is a small probability of violating strict ε-DP. To achieve differential privacy, noise from distributions such as Laplace, Gaussian, or uniform is often added to the data or query results to obscure the influence of any single individual. In the context of SL, DP is often applied to perturb the activations exchanged between the client and server, preventing sensitive information leakage through intermediate representations. Nonetheless, DP methods often entail a trade-off between privacy and model accuracy. Key challenges in these schemes include:

- *Accuracy vs. Noise*: Elevated noise degrades model performance.
- *Scalability*: Tuning DP parameters for high-dimensional data or large-scale models is non-trivial.

Several papers [79, 56, 22] investigate how DP needs to be applied in terms of both achieving privacy and accuracy in various SL setups.

5.1.1.2 Secure Computation:. Cryptographic methods offer formal security guarantees by preventing adversaries from accessing raw data, with each cryptographic scheme preserving different properties. *Homomorphic encryption (HE)* enables computations on encrypted data. Formally, if E represents the encryption function, D the decryption function, and $\circ$ a supported operation (such as addition or multiplication) under HE, then an HE scheme satisfies the property:

$$D(E(x) \circ E(y)) = x \circ y$$

where x and y are plaintext values, and $E(x)$ and $E(y)$ are their corresponding ciphertexts. Each encryption scheme provides different functionalities, and in the context of SL, fully homomorphic encryption is widely preferred due to its ability to support arbitrary polynomial-time computations on encrypted data without requiring decryption.

On the other hand, *function secret sharing (FSS)* is a cryptographic technique that allows a function to be split into n shares distributed among parties, such that the original function can be reconstructed by combining the shares. Below, we present the formal definition of FSS as references in [5].

For $n \in \mathbb{N}$, an n-party FSS scheme with respect to a share output decoder $\text{DEC} = (S_1, \ldots, S_n, R, \text{Dec})$ and a function class $\mathcal{F}$ is a pair of probabilistic polynomial-time algorithms $(\text{Gen}, \text{Eval})$ defined as:

- **Key Generation:** The algorithm $\text{Gen}(1^\lambda, f)$ with a security parameter λ and a function description $f \in \mathcal{F}$, outputs n secret keys $(k_1, \ldots, k_n)$.
- **Evaluation:** Each party i runs $\text{Eval}(i, k_i, x)$, with a party index i, the corresponding key k_i, and an input x. It outputs a value $y_i \in S_i$, representing the party's share of $f(x)$.

For example, in a setting with two semi-honest servers that assumed to not collude, neither server can reconstruct the training function independently, even though together they can compute the intended final result, as $f(x) = f_1(x) \oplus f_2(x)$ where f_1, f_2 are computationally indistinguishable.

The primary objective here is not merely to ensure data privacy, since encryption with well-established cryptographic schemes already provides a rigorous security foundation, but to optimize the application of these strategies for efficiency and practicality, particularly in resource-constrained environments. Khann et al. [38] explore a two-server FSS approach, examining how different secure computation techniques can be adapted for SL.

Other works [35, 61, 57, 41] integrate HE into SL leveraging both cryptographic techniques and adversarial training methodologies to enhance security while maintaining computational feasibility. While these methods maintain model accuracy and provide robust security, their computational overhead can be impractical for resource-constrained environments as discussed in [35], for computational resource-restricted environments additional considerations need to be done to manage the training in a feasible way.

5.1.1.3 Structural and Procedural Modifications:. Defense strategies in this section modify the model architecture or the protocol to limit information leakage and maintain privacy. We provide the details of two subcategories (architectural modifications and protocol modifications) below:

a. Architectural Modifications: Architectural modifications introduce structural changes to SL to prevent reconstruction attacks. Abuadbba et al. [1] investigate the effects of shrinking the cut layer on the effectiveness of FSHA. Building on this, Pham et al. [62] introduce a binarized neural network approach combined with DP to analyze its impact on both security and efficiency. Another important approach leverages *specialized activation functions*, such as the one presented in Equation (10) by Mao et al. [52], which strategically introduce randomness into activation outputs to obscure patterns and minimize the information leakage to potential adversaries.

$$\text{R}^3\text{eLU}(v) = \begin{cases} \max(0, v+z), & \text{with probability } p, \\ 0, & \text{with probability } (1-p). \end{cases} \tag{10}$$

Additionally, mechanisms designed to de-correlate transmitted data have been proposed in [52, 73, 83, 78, 42], demonstrating various techniques to mitigate information leakage while preserving utility.

b. Protocol Modifications: Beyond structural adjustments, refining the protocol that governs data exchange in SL can substantially strengthen privacy. A notable example is SL without local weight sharing (P-SL) [66], which prevents adversaries from leveraging model inversion to reconstruct client data. Instead of sharing model parameters, this approach enables collaborative training without exposing private data. The gradient computation in this modified setting is given by:

$$\nabla L(\text{outputs}, \text{labels}) = \nabla_{u_i, w} L\Big(g_w([z_i, z^{\text{cache}}]), [y_i^{\text{train}}, y^{\text{cache}}]\Big)$$

where $z_i = f_u(x_i)$ represents the smashed data from the current client, and z_{cache} consists of previously cached smashed data, effectively mitigating privacy leakage from individual updates.

An alternative strategy is to *invert the SL setup*, wherein clients hold the labels while the server processes encrypted features, as proposed in [35]. This approach reduces the computational burden on resource-constrained clients while preserving model utility.

Other examples include [66, 44], which explore communication and data-sharing regulations. Such protocol modifications ensure that client-side computations remain efficient while reducing the effectiveness of feature-space hijacking and reconstruction attacks [78, 44]. However, both architecture and protocol level modifications introduce a computational burden on the server, which must be managed to maintain scalability.

While these methods can be empirically effective, they may incur a reduction in model accuracy and often lack formal security proofs, leaving them potentially vulnerable to minor adaptations in attack strategies. Key considerations include:

- *Model Accuracy vs. Privacy*: Structural modifications may compromise performance.
- *Robustness Under Diverse Attacks*: Without formal guarantees, even minor changes in attack tactics might bypass these defenses.

5.1.2. Detection Mechanisms

Detection mechanisms focus on *monitoring* the training process to identify malicious activity *during or after* its occurrence, thereby enabling timely responses such as halting or rolling back updates. These methods are particularly relevant for scenarios involving overtly malicious adversaries who might engage in label-flipping, backdoor insertion, or model hijacking. Two primary detection approaches are employed:

5.1.2.1 Gradient/Update Monitoring:. This method involves analyzing the gradients or updates exchanged between clients and servers to detect abnormal variations. Outlier detection

techniques can highlight significant deviations that may signal an ongoing attack. Several studies have explored gradient anomaly detection to identify adversarial activity in SL [18, 12, 11]. Considering the approach in [18], the detection score DS_n is computed as follows:

$$DS_n = Sig(\lambda_{ds}(\hat{G}_n \cdot \hat{E}_n \cdot \hat{V}_n - \alpha)) \tag{11}$$

where $\hat{G}_n$ represents the adjusted gradient similarity gap, $\hat{E}_n$ is the logarithmic transformation of the polynomial approximation error, and $\hat{V}_n$ quantifies the overlap between same-class and different-class gradients. Sig is a sigmoid activation that normalizes the detection score to the range $[0, 1]$. This approach enables the identification of adversarial gradient manipulations by analyzing deviations in expected gradient behavior.

5.1.2.2 Weight Anomaly Detection:. Some frameworks track model weight evolution, comparing it to normal behavior to detect anomalies and prevent adversarial tampering. Studies such as [69] analyze changes in weight distributions to detect anomalies, particularly those introduced by backdoor attacks, making them effective in identifying malicious training behaviors. One such metric for anomaly detection is the *Rotational Distance* metric introduced in [69], which quantifies the directional change in parameter space between consecutive training iterations:

$$\theta(t) = \arccos\left(\frac{\mathbf{B}_t \cdot \mathbf{B}_{t-1}}{||\mathbf{B}_t||||\mathbf{B}_{t-1}||}\right) \tag{12}$$

where $\mathbf{B}_t$ and $\mathbf{B}_{t-1}$ are the backbone parameters at consecutive training steps, and $\|\cdot\|$ denotes the Euclidean norm. The angular displacement $\theta(t)$ measures the directional shift in model updates. However, backdoor attacks introduce abnormal directional changes in weight updates, causing $\theta(t)$ to deviate from normal training patterns. By averaging pairwise differences in $\theta(t)$ across clients, anomalies can be detected and flagged as potential threats.

Monitoring techniques have been refined to strengthen security while maintaining training integrity and robustness. Rieger et al. [69] propose periodically auditing trained models to validate participant honesty. They highlight the risk of falsely labeling honest participants as malicious, which could disrupt the training process. Their approach ensures that training remains robust while minimizing false positives in adversary detection.

5.2. Defense Constraints

Defense strategies in SL operate under constraints that affect how, and to what degree, they can be deployed: (i) assumptions about adversaries, (ii) requirements for verifying correctness when multiple parties are involved, (iii) the architectural details and training protocol configurations, and (iv) practical limitations in computational power or system availability.

Threat Model Constraints. Alongside the attacks discussed in Section 5.3.1, defense strategies differ in how they address adversaries that may only observe and infer (semi-honest) versus those that actively alter the environment (malicious). For semi-honest scenarios, cryptographic methods [35, 38, 57, 40, 61], data perturbation [79, 64, 77], and architecture-based approaches [62, 1, 52, 73, 44] generally suffice, as they aim to mitigate reconstruction attacks. In malicious settings, verification-based detection strategies [18, 13, 11, 69] are often necessary to detect and prevent adversarial behaviors such as poisoning or backdoor attacks. Balancing these approaches depends on the degree of trust among participants and the trade-off between privacy and performance overhead.

Multi-Party Verification Constraints. When multiple clients or servers are involved, defenses may need enhanced validation mechanisms to differentiate between benign and malicious updates, as demonstrated in [13, 11, 18]. Some methods [69, 42] rely on cross-comparisons among clients, a trusted third party, or a server, which increases communication rounds and latency. In large-scale or bandwidth-limited environments, frequent verification may be impractical. Leveraging powerful nodes or third parties can mitigate some of these issues but introduces additional trust assumptions.

Architectural Constraints. Certain defenses [62, 1] depend on the model's partitioning and the number of layers retained on the client side [62] or specialized activation functions [52], can limit information leakage but may also reduce flexibility or performance on complex tasks. Architectural considerations become increasingly crucial when striving for a generalized approach that minimizes reliance on specific empirical setups.

Computational and Operational Constraints. Practical deployments of SL, especially on edge or mobile devices, limit how often encryption, extra communication rounds, or gradient checks can be performed. Heavy cryptographic schemes may be infeasible, and much of the work in this domain focuses on improving computational efficiency [35, 57, 40]. Additionally, defenses that assume continuous client availability are prone to disruption if participants drop out or arrive late, as considered in [69]. Some training modifications also limit the flow of information during learning [66]. These operational constraints highlight the need for strategies that remain robust under unpredictable network conditions and resource limitations.

5.3. Defense Effectiveness

Defense effectiveness in SL is evaluated based on several key factors, including applicability to different attack types, assumed client-server setups, computational constraints, and tested model architectures. The alignment between defense strategies and attack vectors is crucial, as protection-based approaches such as cryptographic techniques, DP, and architectural modifications focus on preventing information leakage, while detection-based methods such as gradient monitoring and anomaly detection aim to identify and halt adversarial behavior.

5.3.1. Attack Coverage and Defense Alignment

Each defense mechanism addresses specific adversarial threats in SL. *Data perturbation defenses* [1, 79, 64, 77], including DP and feature decorrelation, are effective against *label inference and reconstruction attacks*, making it more difficult

for adversaries to extract information from gradients or intermediate representations. However, these approaches often introduce a trade-off in accuracy. *Secure computation techniques* [35, 57, 41, 61, 38], such as HE and FSS, offer robust privacy guarantees by allowing encrypted computations.

Architectural modifications[62, 1, 66, 44, 83, 52, 82, 73, 78, 42], including cut-layer adjustments and novel activation functions, significantly mitigate *feature-space hijacking and reconstruction attacks*, but their effectiveness depends on the model's depth and complexity.

Detection mechanisms are crucial for identifying adversarial actions during training. *Gradient anomaly detection* [11, 13, 18] is effective against *poisoning and hijacking attacks* by analyzing inconsistencies in the updates exchanged between clients and servers. *Model weight anomaly detection* [69] is particularly useful against *backdoor attacks*, as it can identify hidden manipulations within the model weights by monitoring deviations from expected parameters. The effectiveness of these mechanisms relies on accurately profiling normal training behavior, and they may be vulnerable to adversaries who subtly manipulate updates to evade detection.

5.3.2. Client-Server Setup and Deployment Scenarios

The client-server setup influences the feasibility and effectiveness of defenses. In *single-client SL (SCSL)*, cryptographic defenses such as HE and FSS are more feasible, as encryption and decryption overhead is limited to a single participant. Architectural modifications are also easier to apply, as there is no need for cross-client consistency.

In *multi-client SL (MCSL)*, on the other hand, *client-side gradient monitoring and anomaly detection* become more relevant, as adversarial participants may be disguised among honest clients. *Data perturbation techniques*, including DP, require careful tuning in MCSL, as variations in data distributions may affect privacy-utility trade-offs. In addition, DP alone are insufficient in terms of defense as discussed earlier (Section 4.3). *Secure computation methods* face scalability challenges, as the overhead increases with an increasing number of clients. In MCSL, *architectural modifications* such as split-layer depth optimization and secure aggregation become more practical, as computational burdens are distributed across multiple parties. Lastly, communication latency and encryption overhead remain key concerns for cryptographic techniques.

5.3.3. Model and Dataset Considerations

Defense effectiveness also depends on the complexity of the model architecture and the dataset's properties. In *lightweight models* such as convolutional neural networks for image classification, *DP and gradient perturbation* are effective [64, 77], as small distortions in features do not drastically impact performance. Cryptographic methods such as HE remain feasible in small-scale models but introduce significant latency [35, 39]. In *deep architectures* such as ResNet and transformers, *secure computation techniques* struggle due to their high computational complexity, while *architectural modifications* [1, 62, 52] become more critical to mitigate feature-space hijacking. *Detection-based methods*, such as gradient anomaly analysis, are highly dependent on the model architecture and training setup, as the information carried in gradients is significantly influenced by these factors. For instance, as discussed in [11], larger gradients may suffer from the curse of dimensionality, since these methods rely on distance and correlation analysis among data.

6. Key Observations and Takeaways

We outline key observations derived from our analysis on SL attack and defenses below:

- **Architecture:** The feasibility of attacks and defenses is heavily influenced by the architectural design (e.g., VanSL, USL) and data partitioning strategies (horizontal vs. vertical). We observe that most attack research focuses on VanSL, which is highly vulnerable to feature-space hijacking and model inversion due to the exposure of smashed data at a single cut layer. In contrast, USL and SFL face increased label exposure risks due to multi-hop data transmissions. Similarly, the majority of defenses focus on the VanSL setting (Figure 7). Furthermore, we observe that No-Label SL (NLSL) and Multi-hop SL (MHSL) are the least explored variants in the literature despite their distinct privacy and trust characteristics. NLSL enhances label privacy by keeping labels on the client, requiring thorough analysis to assess its resistance to inference attacks targeting label information via gradients or activations. MHSL, with its chained computation across multiple entities, has received minimal attention in the literature, and its unique multi-party computation and trust assumptions remain inadequately characterized. These observations directly address *RQ1*, which seeks to understand how adversarial objectives can be systematically taxonomized in relation to architectural vulnerabilities in SL. Our findings underscore the need to extend taxonomies beyond VanSL to capture under-explored variants with distinct structural vulnerabilities.
- **Semi-Honest Adversaries:** Our taxonomy reveals a critical reliance on the semi-honest adversarial model, where adversaries passively follow the protocol. While this simplifies threat modeling, it underestimates real-world risks, as attacks like label flipping, gradient manipulation, or collusion exceed the assumptions of this model, rendering many defenses ineffective against stronger adversaries per *RQ3*.
- **Generalization of Attack Mechanisms:** Several core attack strategies, especially those that exploit the cut layer bottleneck, such as latent representation reconstruction [59, 52] and property inference based on intermediate activations [59], demonstrated effectiveness across a variety of SL models and datasets. This suggests that the act of splitting and transmitting intermediate representations introduces consistent privacy vulnerabilities that are not easily mitigated by simply changing model architecture or task. These findings respond to *RQ2*, which seeks to understand the generalizability and stealth of SL attack mechanisms. The persistence of these attacks across different conditions suggests an architectural invariance in SL's threat surface. This highlights both a concern and an opportunity: to develop model-

agnostic defenses that reduce information leakage in representations z and gradients ∇ as discussed in Section 4.1.2, for example, via dimensionality reduction.

- **Inherent Risk of the Cut Layer:** The interface between the client-side and server-side models, where z or ∇ are exchanged, consistently emerges as a critical attack surface in SL. Multiple attacks, including FSHA [59], model inversion via gradients [10], GAN-based input reconstruction [91], and various label or property inference strategies [37, 59, 10], exploit this specific interface to extract sensitive information. This highlights the need to prioritize specifically the security of the cut layer. Moreover, it underscores that defenses focused solely on obfuscating raw inputs, such as local DP (see Section 4.3), are insufficient, as critical information is often preserved and leaked through z and ∇. Effective SL defenses must directly mitigate the information content traversing this unavoidable interface between model partitions, which directly relates to *RQ2's* examination of attack vectors and vulnerable interfaces.
- **Reliance on Strong Adversarial Capabilities:** Many prominent SL attack studies assume strong adversarial capabilities, such as full or partial model access [59, 91] or access to auxiliary datasets aligned with the training distribution [92, 32, 89]. While useful for exposing theoretical vulnerabilities, these assumptions often diverge from practical deployment scenarios, where adversaries may have limited knowledge, access, or resources. In addressing *RQ3*, we systematically differentiate between strong and constrained adversarial assumptions across attack studies, enabling more accurate threat modeling and highlighting the need for context-aware, practically-deployable defenses.
- **Leakage Persistence Beyond Gradients:** Studies have demonstrated that intermediate activations exchanged during the forward pass can leak sensitive information, enabling label inference [48, 96] and data reconstruction [52, 59] even in the absence of gradient information. This critical finding shows that SL remains vulnerable even during inference-only deployments or when gradient leakage is mitigated through secure aggregation or other protective mechanisms. This emphasizes the need for defenses targeting information leakage in z —such as its semantic consistency or correlation with inputs— beyond just the backward pass, addressing *RQ1* on the nature and sources of information leakage in SL.
- **Privacy–Utility–Overhead Trade-off:** SL defenses show a clear three-way trade-off between privacy, model utility (e.g. accuracy), and computational or communication overhead. Methods range from lightweight approaches with weaker guarantees to heavy cryptographic schemes with high costs; while DP offers a tunable middle ground whose optimal balance is still application-specific and challenging to achieve. Homomorphic Encryption (HE) and Function Secret Sharing (FSS) offer strong theoretical privacy by enabling computation on encrypted data. However, their high computational and bandwidth overheads restrict practical use to simple models or scenarios, where privacy outweighs performance. This addresses *RQ4* by discussing the place of cryptographic techniques in the landscape of defense techniques. The three-way trade-off is addressing *RQ5*, which discusses the suitability, constraints, and potential optimization of defenses for specific tasks, and also connects to *RQ6* regarding what constitutes a *successful* defense in practice, often involving a balance of these factors.
- **Architectural Changes:** Modifying SL —by shifting the cut layer or shrinking the shared 'smashed data'— can reduce exposure and increase attack difficulty, but studies agree that these changes alone rarely ensure full or provable privacy, serving best as complements to stronger defenses. This observation aligns with *RQ4's* investigation into the strengths and limitations of current defense strategies in mitigating information leakage. Architectural changes are a good fit for situations where one does not have much computing power or when one wants to add a basic, extra layer of security alongside stronger methods.
- **Confidentiality & Integrity:** Most SL defenses target two goals: confidentiality (protecting data, labels, or attributes) and integrity (preventing model manipulation with monitoring or robust aggregation). Availability remains underexplored in the reviewed literature. This distinction between defense goals directly informs *RQ6* —how techniques prioritize confidentiality vs. integrity— and supports the taxonomy outlined in *RQ4*; notably, detection strategies, crucial for identifying active threats like poisoning or backdoors (though reliant on robust baselines and unable to address passive leakage), fall under this discussed class and thus also address aspects of *RQ4*.
- **Hybrid Defenses:** While layering defense techniques may seem promising, the literature highlights complex interactions—such as applying DP to encrypted data or combining noise with architectural constraints, which can increase overhead, obscure true privacy guarantees, or introduce new vulnerabilities. These findings, central to *RQ5*, emphasize that effective composition requires careful joint analysis, as naive stacking is often inadequate.
- **Prominence of Studied Defenses:** The literature shows DP as the most extensively studied approach for enhancing confidentiality in SL, followed by HE. In contrast, detection research focuses primarily on gradient analysis. Highlighting these commonly used techniques —DP, HE, and gradient analysis— directly informs *RQ4* on prevalent SL defense strategies.

7. Open Research Directions

Building on our key observations, we outline the key open research directions below:

- **Advanced Attack and Defense Strategies:** A key research direction emerging in response to *RQ1* involves extending adversarial taxonomies and corresponding defense mechanisms to SL variants beyond the conventional VanSL setting. Alternative SL variants such as USL, Hybrid-SL, NLSL, and MHSL exhibit distinct architectural patterns (e.g., multi-hop communication, hybrid cut points) and data handling procedures, which introduce previously unaddressed attack surfaces. As outlined in Section 4 and Section 5, these non-

standard configurations may necessitate variant-specific defense strategies that are not adequately covered by existing VanSL-centric approaches. Alongside exploring novel approaches, techniques from the broader PPML literature could be adapted to SL when there are matching problem structures. For instance, adaptive differential privacy used in federated learning [17, 75, 7] could be incorporated to dynamically adjust noise levels based on model updates. This adjustment could help balance the trade-off between model accuracy and privacy by reducing noise in scenarios where updates pose a lower privacy risk.

- **Broadening Threat Models Beyond Semi-Honest Adversaries:** Future research should focus on developing a progressive adversarial framework that evaluates defenses across a spectrum of adversarial strengths, from semi-honest actors to fully malicious colluding parties. SL systems are especially vulnerable to adversaries capable of poisoning gradients, manipulating activations, or tampering with training states. We advocate for hybrid evaluation protocols that integrate empirical testing with theoretical analysis to assess robustness. Furthermore, adopting formal models from cryptographic literature (e.g., Byzantine threat models [15], covert adversaries [3], or adaptive adversaries [2]) can help bridge the current gap between theoretical security guarantees and practical adversarial capabilities, thus advancing the scope of *RQ3* on evaluating the realism and comprehensiveness of adversarial assumptions.
- **Toward Uncertainty-Aware Defense Design:** A promising research direction is the development of uncertainty-aware defenses via entropy analysis of intermediate representations in SL systems. Low entropy in smashed data —often caused by deterministic or sparse activations— correlates with increased vulnerability to inference attacks, as it signals reduced uncertainty exploitable by adversaries. Operations like ReLU and max-pooling compress representational diversity, aligning features closely with input semantics and elevating privacy risk. In contrast, randomized activations and dropouts introduce beneficial uncertainty that hinders inference. Formalizing these effects using information-theoretic tools like mutual information and representation entropy could quantify the trade-off between expressivity and leakage. Additionally, comparing differential entropy before and after the cut layer may help identify optimal cut points that balance utility and privacy. Entropy-based metrics thus offer a principled foundation for designing more privacy-resilient SL architectures, contributing to *RQ2* on structural factors that shape the effectiveness of inference attacks in SL.
- **Evaluating Practical Threats Under Realistic Conditions:** In response to *RQ3*, future research should prioritize the development and evaluation of attacks under more constrained and realistic conditions. This includes black-box settings, limited or mismatched auxiliary data, bounded query or compute budgets, and scenarios involving partial or probabilistic knowledge of the target model. Crucially, quantifying how attack effectiveness degrades under practical constraints is vital to building accurate and actionable threat models for real-world SL deployments.
- **Advancing Holistic Monitoring Strategies:** Future research should prioritize proactive, multi-layered defenses to address SL's broad attack surface: a direction already emerging in recent work, as noted in Section 5.1. Integrating anomaly detection (e.g., gradient or behavior-based) [11] with insights from federated learning —such as techniques developed to achieve secure aggregation for verifying client updates [36]— can enhance detection capabilities without significant overhead. Additionally, defenses against side-channel threats (e.g., [8]) and secure inference techniques can strengthen robustness against covert attacks. Hybrid approaches may be necessary —combining secure computation methods (for confidentiality of client data) with monitoring (for learning integrity)— to realize a truly holistic defense strategy in SL. This means that monitoring mechanisms should be integrable with other defense techniques.
- **Instance Encoding and Potential Threats:** Intermediate representations in SL —often referred to as smashed data— can be viewed as encodings that have the potential to leak sensitive information, even without full input reconstruction. This issue is known as the *instance encoding problem* [6, 50]. Carlini et al.[6] reframed the PPML problem as instance encoding, analyzing the statistical underpinnings of privacy in ML, comparing techniques, and offering bounds and empirical results under specific attacks. Maeng et al.[50] introduced a Fisher information based framework to quantify and bound leakage in PPML, also leveraging the instance encoding perspective. Given the alignment in goals and definitions, this concept provides valuable insights for improving the security of SL systems.
- **Verifiable Training and Inference:** Ensuring the verifiability of training and inference in SL, alongside privacy, is another crucial objective. Zero knowledge (ZK)-proofs enable parties to prove correct computation without leaking unauthorized information, offering a promising means to verify the correctness of training in PPML. This capability is vital where malicious participants alter training data, introduce backdoors, or poison the training process. Peng et al. [60] survey the use of ZK-proofs in machine learning, categorizing approaches into verifiable training, testing, and inference, and proposing an abstract framework for their application in ML contexts, which can be beneficial for SL settings. Efficiency would constitute a crucial concern when ZK-proofs are employed.
- **Exploring Hardware-Based Security Solutions:** Hardware-based solutions, such as trusted execution environments (e.g., [31]), are under-explored and show promise for enhancing security and privacy in SL.
- **Resilience to Distributed System Faults** Ensuring the training process remains robust and continues effectively despite common distributed system faults, such as client dropouts, late-arrivals, and potential communication errors. Strategies focus on maintaining training integrity and progress even when parts of the system fail.

8. Related Work

Several Systematization of Knowledge (SoK) papers have examined privacy-preserving methodologies across collaborative learning paradigms. Podschwadt et al. [67] presented a systematization of deep learning to preserve privacy utilizing HE, focusing on the computational complexities and practical limitations of HE-based solutions. Ng and Chow [55] conducted an extensive SoK examining cryptographic approaches to privacy preservation in deep learning. Mansouri et al. [51] systematically analyzed secure aggregation techniques for FL, categorizing encryption-based and MPC-based approaches while evaluating their performance, scalability, and resilience against adversarial attacks.

While FL and SL share distributed learning characteristics, SL presents distinct challenges arising from its architectural design, particularly in the transmission of intermediate activations between participants. Conventional secure aggregation methods developed for FL cannot be directly applied to SL without substantial modifications, as SL's vulnerabilities emerge from unique adversarial capabilities, including feature-space leakage and inference attacks. In examining SL-specific security concerns, Pham and Chilamkurti [63] surveyed data leakage threats, identifying gradient leakage, label inference, and feature reconstruction attacks. Hu et al. [29] conducted a review and experimental evaluation of SL, highlighting the variability in SL paradigms concerning cut-layer selection, model aggregation, and label sharing. While these works provide highly valuable empirical insights, we uniquely and systematically categorize security and privacy challenges in SL, establishing a novel formal taxonomy of attack vectors and mitigation techniques, analyzing their effectiveness and limitations.

9. Conclusion

We systematically explored the security and privacy landscape of Split Learning (SL) by categorizing various attack strategies and defense mechanisms. We presented a novel taxonomy of attacks and defenses in SL, categorizing them along: (i) strategies, (ii) constraints, and (iii) effectiveness. While SL presents a promising framework for distributed/outsourced machine learning, its inherent vulnerabilities -—ranging from data reconstruction and label inference to adversarial manipulation—- underscore the need for robust countermeasures. Existing defenses –including cryptographic techniques, differential privacy, and architectural modifications– often introduce trade-offs in computational efficiency and scalability. Employing the key observations and takeaways presented in this paper, future research can enhance the resilience of SL, making it a viable solution for secure collaborative learning.

Acknowledgements

We acknowledge TÜBİTAK (the Scientific and Technological Research Council of Türkiye) project 124N941. The authors utilized ChatGPT-4o [58] to refine the text in Sections 1, 4, and 5 for improving text, shortening sentences, correcting typos and grammatical errors to enhance readability and clarity.

References

[1] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal. Can we use split learning on 1d cnn models for privacy preserving training? In *ASIACCS*, 2020.

[2] R. Arora, O. Dekel, and A. Tewari. Online bandit learning against an adaptive adversary: from regret to policy regret. *arXiv:1206.6400*, 2012.

[3] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 2010.

[4] Y. Bai, Y. Chen, H. Zhang, W. Xu, H. Weng, and D. Goodman. {VILLAIN}: Backdoor attacks against vertical split learning. In *USENIX Security*, 2023.

[5] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT*. Springer, 2015.

[6] N. Carlini, S. Deng, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, S. Song, A. Thakurta, and F. Tramèr. Is private learning possible with instance encoding? 2021.

[7] Z. Chen, H. Zheng, and G. Liu. Awdp-fl: An adaptive differential privacy federated learning framework. *Electronics*, 2024.

[8] E. Debenedetti, G. Severi, N. Carlini, C. A. Choquette-Choo, M. Jagielski, M. Nasr, E. Wallace, and F. Tramèr. Privacy side channels in machine learning systems. In *USENIX Security*, 2024.

[9] C. Dwork. Differential privacy. In *ICALP*, 2006.

[10] E. Erdoğan, A. Küpçü, and A. E. Çiçek. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *ACM WPES*, 2022.

[11] E. Erdogan, U. Teksen, M. S. Celiktenyildiz, A. Kupcu, and A. E. Cicek. Splitout: Out-of-the-box training-hijacking detection in split learning via outlier detection. In *CANS*, 2024.

[12] E. Erdoğan, A. Küpçü, and A. E. Çiçek. Splitguard: Detecting and mitigating training-hijacking attacks in split learning. In *WPES*, 2022.

[13] E. Erdoğan, U. Tekşen, M. S. Çeliktenyıldız, A. Küpçü, and A. E. Çiçek. Defense mechanisms against training-hijacking attacks in split learning. *IEEE TKDE*, 2023.

[14] M. Fan, C. Chen, C. Wang, W. Zhou, and J. Huang. On the robustness of split learning against adversarial attacks. In *ECAI*. 2023.

[15] M. Fang, X. Cao, J. Jia, and N. Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *USENIX Security*, 2020.

[16] C. Fu, J. Zhang, T. Zhu, W. Zhou, and P. S. Yu. Label inference attacks against vertical federated learning. In *USENIX Security*, 2022.

[17] J. Fu, Z. Chen, and X. Han. Adap dp-fl: Differentially private federated learning with adaptive noise, 2022.

[18] J. Fu, X. Ma, B. B. Zhu, P. Hu, R. Zhao, Y. Jia, P. Xu, H. Jin, and D. Zhang. Focusing on pinocchio's nose: A gradients scrutinizer to thwart split-learning hijacking attacks using intrinsic attributes. In *NDSS*, 2023.

[19] S. Gajbhiye, P. Singh, and S. Gupta. Data poisoning attack by label flipping on splitfed learning. In *RTIP2R*, 2022.

[20] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, 2015.

[21] Y. Gao, M. Du, X. Zhang, and Y. Xiang. Combined federated and split learning in edge computing: Taxonomy and open issues. *Sensors*, 2022.

[22] G. Gawron and P. Stubbings. Feature space hijacking attacks against differentially private split learning. *arXiv:2201.04018*, 2022.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, 2014.

[24] A. Gretton, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, K. Fukumizu, and B. K. Sriperumbudur. Optimal kernel choice for large-scale two-sample tests. *NIPS*, 2012.

[25] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents, 2018.

[26] Y. He, C. Hu, Y. Pu, J. Chen, and X. Li. Advusl: Targeted adversarial attack against u-shaped split learning. In *IEEE MASS*, 2024.

[27] Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In *ACSAC*, 2019.

[28] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *ACM SIGSAC*, 2017.

[29] Z. Hu, T. Zhou, B. Wu, C. Chen, and Y. Wang. A review and experimental evaluation on split learning. *Future Internet*, 2025.

[30] H. Huang, X. Li, and W. He. Pixel-wise reconstruction of private data in split federated learning. In *ICICS*, 2023.

[31] W. Huang, Y. Wang, A. Cheng, A. Zhou, C. Yu, and L. Wang. A fast, performant, secure distributed training framework for llm. In *ICASSP*, 2024.

[32] A. T. Z. Ismail and R. M. Shukla. Analyzing the vulnerabilities in splitfed learning: Assessing the robustness against data poisoning attacks. *arXiv:2307.03197*, 2023.

[33] A. T. Z. Ismail and R. M. Shukla. Analyzing the vulnerabilities in splitfed learning: Assessing the robustness against data poisoning attacks, 2023.

[34] P. Joshi, C. Thapa, S. Camtepe, M. Hasanuzzaman, T. Scully, and H. Afli. Performance and information leakage in splitfed learning and multi-head split learning in healthcare data and beyond. *Methods and Protocols*, 2022.

[35] H. I. Kanpak, A. Shabbir, E. Genç, A. Küpçü, and S. Sav. Cure: Privacy-preserving split learning done right, 2024.

[36] F. Karakoç, A. Küpçü, and M. Önen. Fault tolerant and malicious secure federated learning. In *CANS*, 2024.

[37] S. Kariyappa and M. K. Qureshi. Exploit: Extracting private labels in split learning. In *SaTML*. IEEE, 2023.

[38] T. Khan, M. Budzys, and A. Michalas. Make split, not hijack: Preventing feature-space hijacking attacks in split learning. In *SACMAT*, 2024.

[39] T. Khan, K. Nguyen, and A. Michalas. A more secure split: Enhancing the security of privacy-preserving split learning. In *AsiaCCS*. Tampere University, 2023.

[40] T. Khan, K. Nguyen, and A. Michalas. Split ways: Privacy-preserving training of encrypted data using split learning. In *arXiv:2301.08778, 2023*, 2023.

[41] T. Khan, K. Nguyen, A. Michalas, and A. Bakas. Love or hate? share or split? privacy-preserving training using split learning and homomorphic encryption, 2023.

[42] S. A. Khowaja, I. H. Lee, K. Dev, M. A. Jarwar, and N. M. F. Qureshi. Get your foes fooled: Proximal gradient split learning for defense against model inversion attacks on iomt data. *IEEE TNSE*, 2024.

[43] M. Kohankhaki, A. Ayad, M. Barhoush, and A. Schmeink. Detecting data poisoning in split learning using intraclass-distance inflated loss. In *IEEE GC Wkshps*, 2023.

[44] J. Li, A. S. Rakin, X. Chen, Z. He, D. Fan, and C. Chakrabarti. Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning. In *CVPR*, 2022.

[45] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang. Label leakage and protection in two-party split learning. *arXiv:2102.08504*, 2018.

[46] Z. Li, S. Si, J. Wang, and J. Xiao. Federated split bert for heterogeneous text classification, 2022.

[47] Z. Li, C. Yan, X. Zhang, G. Gharibi, Z. Yin, X. Jiang, and B. A. Malin. Split learning for distributed collaborative training of deep learning models in health informatics. In *AMIA Annu. Symp. Proc*, 2024.

[48] J. Liu, X. Lyu, Q. Cui, and X. Tao. Similarity-based label inference attack against training and inference of split learning. *IEEE TIFS*, 2024.

[49] M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *PMLR*, 2015.

[50] K. Maeng, C. Guo, S. Kariyappa, and G. E. Suh. Bounding the invertibility of privacy-preserving instance encoding using fisher information. *NeurIPS*, 2023.

[51] M. Mansouri, M. Önen, W. B. Jaballah, and M. Conti. Sok: Secure aggregation based on cryptographic schemes for federated learning. *PoPETs*, 2023.

[52] Y. Mao, Z. Xin, Z. Li, J. Hong, Q. Yang, and S. Zhong. Secure split learning against property inference, data reconstruction, and feature space hijacking attacks. In *ESORICS*, 2023.

[53] M. Nasr, R. Shokri, and A. Houmansadr. Machine learning with membership privacy using adversarial regularization. In *ACM SIGSAC*, 2018.

[54] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*, 2019.

[55] L. K. Ng and S. S. Chow. Sok: cryptographic neural-network computation. In *IEEE S&P*, 2023.

[56] K. T. P. Ngoc Duy Pham and N. Chilamkurti. Enhancing accuracy-privacy trade-off in differentially private split learning, 2024.

[57] K. Nguyen, T. Khan, and A. Michalas. Split without a leak: Reducing privacy leakage in split learning. In *SecureComm*, 2025.

[58] OpenAI. Gpt-4o: Multimodal ai model.

[59] D. Pasquini, G. Ateniese, and M. Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *ACM CCS*, 2021.

[60] Z. Peng, T. Wang, C. Zhao, G. Liao, Z. Lin, Y. Liu, B. Cao, L. Shi, Q. Yang, and S. Zhang. A survey of zero-knowledge proof based verifiable machine learning, 2025.

[61] G.-L. Pereteanu, A. Alansary, and J. Passerat-Palmbach. Split he: Fast secure inference combining split learning and homomorphic encryption, 2022.

[62] N. D. Pham, A. Abuadbba, Y. Gao, T. K. Phan, and N. Chilamkurti. Binarizing split learning for data privacy enhancement and computation reduction, 2022.

[63] N. D. Pham and N. Chilamkurti. Data leakage threats and protection in split learning: A survey. In *ICEA*, 2023.

[64] N. D. Pham, K. T. Phan, and N. Chilamkurti. Enhancing accuracy-privacy trade-off in differentially private split learning. *IEEE TIFS*, 2024.

[65] N. D. Pham, T. K. Phan, A. Abuadbba, Y. Gao, D. Nguyen, and N. Chilamkurti. Split learning without local weight sharing to enhance client-side data privacy. *arXiv:2212.00250*, 2022.

[66] N. D. Pham, T. K. Phan, A. Abuadbba, Y. Gao, V.-D. Nguyen, and N. Chilamkurti. Split Learning without Local Weight Sharing To Enhance Client-side Data Privacy . *IEEE TDSC*, (01):1–13, Apr. 5555.

[67] R. Podschwadt, D. Takabi, and P. Hu. Sok: Privacy-preserving deep learning with homomorphic encryption. *arXiv:2112.12855*, 2021.

[68] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar. Split learning for collaborative deep learning in healthcare. *arXiv:1912.04966*, 2019.

[69] P. Rieger, A. Pegoraro, K. Kumari, T. Abera, J. Knauer, and A.-R. Sadeghi. Safesplit: A novel defense against client-side backdoor attacks in split learning. In *NDSS*, 2025.

[70] H. R. Roth, A. Hatamizadeh, Z. Xu, C. Zhao, W. Li, A. Myronenko, and D. Xu. Split-u-net: Preventing data leakage in split learning for collaborative multi-modal brain tumor segmentation, 2022.

[71] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 1992.

[72] M. D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 2011.

[73] A. Singh, A. Chopra, V. Sharma, E. Garza, E. Zhang, P. Vepakomma, and R. Raskar. Disco: Dynamic and invariant sensitive channel obfuscation for deep neural networks, 2021.

[74] B. Tajalli, O. Ersoy, and S. Picek. On feasibility of server-side backdoor attacks on split learning. In *IEEE SPW*, 2023.

[75] M. Talaei and I. Izadi. Adaptive differential privacy in federated learning: A priority-based approach. *arXiv:2401.02453*, 2024.

[76] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun. Splitfed: When federated learning meets split learning. In *AAAI*, 2022.

[77] T. Titcombe, A. J. Hall, P. Papadopoulos, and D. Romanini. Practical defences against model inversion attacks for split neural networks. In *ICLR Workshop on DPML*, 2021.

[78] V. Turina, Z. Zhang, F. Esposito, and I. Matta. Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures. In *CLOUD*, 2021.

[79] P. Vepakomma, J. Balla, and R. Raskar. Privatemail: Supervised manifold learning of deep features with differential privacy for image retrieval, 2021.

[80] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar. Reducing leakage in distributed deep learning for sensitive health data. *arXiv:1812.00564*, 2019.

[81] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data, 2018.

[82] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar. Nopeek: Information leakage reduction to share activations in distributed deep learning, 2020.

[83] X. Wan, J. Sun, S. Wang, L. Chen, Z. Zheng, F. Wu, and G. Chen. Pslf: Defending against label leakage in split learning. In *ACM CIKM*, 2023.

[84] Y. Wang, C. Zhang, Z. Zheng, J. Wang, and X. Li. Stitchable split learning assisted multi-uav systems. *IEEE Open J. Comput. Soc*, 2024.

[85] X. Wu, H. Yuan, X. Li, J. Ni, and R. Lu. Evaluating security and robustness for split federated learning against poisoning attacks. *IEEE TIFS*, 2024.

[86] S. Xie, X. Yang, Y. Yao, T. Liu, T. Wang, and J. Sun. Label inference attack against split learning under regression setting. *arXiv:2301.07284*, 2023.

[87] X. Xu, M. Yang, W. Yi, Z. Li, J. Wang, H. Hu, Y. Zhuang, and Y. Liu. A stealthy wrongdoer: Feature-oriented reconstruction attack against split learning. In *CVPR*, 2024.

[88] X. Yang, J. Sun, Y. Yao, J. Xie, and C. Wang. Differentially private label protection in split learning. *arXiv:2203.02073*, 2022.

[89] F. Yu, L. Wang, B. Zeng, K. Zhao, Z. Pang, and T. Wu. How to backdoor split learning. *Neural Networks*, 2023.

[90] F. Yu, B. Zeng, K. Zhao, Z. Pang, and L. Wang. Chronic poisoning: Backdoor attack against split learning. In *AAAI*, 2024.

[91] B. Zeng, S. Luo, F. Yu, G. Yang, K. Zhao, and L. Wang. Gan-based data reconstruction attacks in split learning. *Neural Networks*, 2025.

[92] L. Zhang, X. Gao, Y. Li, and Y. Liu. Functionality and data stealing by pseudo-client attack and target defenses in split learning. *IEEE TDSC*, 2024.

[93] K. Zhao, X. Chuo, F. Yu, B. Zeng, Z. Pang, and L. Wang. Splitaum: Auxiliary model-based label inference attack against split learning. *IEEE TNSM*, 2024.

[94] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. *arXiv:1806.00582*, 2018.

[95] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *NeurIPS*, 2019.

[96] X. Zhu, X. Luo, Y. Wu, Y. Jiang, X. Xiao, and B. C. Ooi. Passive inference attacks on split learning via adversarial regularization. *arXiv:2310.10483*, 2023.

Supplementary Material

A. Additional Split Learning Variants

Here, we explain the remaining SL variants, namely multi-hop and no-label split learning.

Multi-hop Split Learning (MHSL): MHSL extends the SL concept to multiple parties, creating a chain of computation across multiple entities. The model is divided into several segments ($f_{c_1}, f_{s_1}, f_{s_2}, ..., f_{c_n}$) where the subscript denotes the entity processing that segment. The client with input x begins by computing ($z_{c_1} = f_{c_1}(x)$) and sends z_{c_1} to the first server. Each subsequent server i computes $z_{s_i} = f_{s_i}(z_{s_{i-1}})$ and forwards it to the next entity. The final client computes the output $\hat{y} = f_{c_n}(z_{s_{n-1}})$ and calculates the loss. During backpropagation gradients flow in the reverse direction through the chain. While the ordering of clients and servers in MHSL is flexible, there are practical constraints. A client must process the first model segment to handle the x, and the last entity whether a client or a server must compute the loss. Intermediate entities can be either servers or clients, depending on computational needs. For example, in a medical collaboration setting, hospitals can process the initial and final model segments while cloud servers handle intermediate computations. In contrast, in a business forecasting scenario, multiple companies may preprocess data before sending it to a server for final inference. This flexibility allows MHSL to adapt to various distributed learning scenarios while ensuring privacy and scalability.

No-Label Split Learning (NLSL): NLSL is designed to keep both x and y private to the client. In NLSL, the client processes f_c to compute z_c and sends it to the server. The server computes the forward pass through the remaining layers f_s to generate $\hat{y}$. However, rather than calculating the loss on the server, the server sends $\hat{y}$ back to the client. The client then computes the loss $L(y, \hat{y})$ using the true labels y and calculates the gradient ($\nabla_{\hat{y}} = \frac{\partial L}{\partial \hat{y}}$). The client sends $\nabla_{\hat{y}}$ back to the server, which begins the backpropagation process by computing ($\nabla_{z_c} = \frac{\partial L}{\partial z_c}$) and sending it to the client. Finally, the client updates the parameters of f_c using the received gradients. Similar to USL, NLSL ensures that the client is responsible for computing the loss, preventing the server from having direct access to y. However, unlike USL, where the client also processes the final layers after receiving intermediate representations from the server, in NLSL, the server completes the entire forward pass, ensuring that only the client handles label-related computations, offering stronger privacy guarantees.

B. Details of Data Reconstruction Attacks

Data reconstruction attacks in split learning exploit smashed data z_c to infer private client inputs. They are primarily categorized into (i) Feature Space Hijacking Attack (FSHA), (ii) Model Inversion Attack, (iii) Functionality Stealing, (iv) Generative Adversarial Network (GAN), and (v) Feature Reconstruction techniques. Each technique exploits distinct aspects of the model behavior to compromise training and data privacy as explained below:

i. Feature Space Hijacking Attack (FSHA):. A prominent data reconstruction attack is FSHA [59, 22], which allows a malicious server to reconstruct the private data in a VanSL setup. FSHA consists of three key components: (i) Pilot network ($\tilde{f}_c$): dynamically defines the target feature space $\tilde{z}_c$ and is responsible for mapping between raw input x and $\tilde{z}_c = \tilde{f}_c(x)$ for the client network. (ii) Inverse network ($\tilde{f}_c^{-1}$): trained to approximate the inverse function of $\tilde{f}_c$, allowing the malicious server to reconstruct x from $\tilde{z}_c$, and (iii) Discriminator (D): an adversarially-trained network that indirectly guides to learn the mapping between x and $\tilde{z}_c$. FSHA follows a two-step adversarial training process. In the first step, the server begins by sampling a batch from public dataset x_{pub} to train $\tilde{f}_c$ and $\tilde{f}_c^{-1}$, ensuring the networks converge by minimizing the reconstruction loss:

$$L_{\tilde{f}_c, \tilde{f}_c^{-1}} = d(\tilde{f}_c^{-1}(\tilde{f}_c(x_{pub}), x_{pub}) \tag{13}$$

where d is a suitable distance function, such as Mean Squared Error (MSE). In the second step, the server adversarially trains the D loss to distinguish between $\tilde{z}_c$ from the one induced from the z_c:

$$L_D = \log(1 - D(\tilde{z}_c)) + \log(D(z_c)) \tag{14}$$

After each local training step for D, the malicious server can then train the network by forging the gradient using D to reconstruct an adversarial loss function for f_c: $L_{f_c} = \log(1 - D(z_c))$. After adversarial training, the server can reconstruct the client's private input using: $\tilde{x} = \tilde{f}_c^{-1}(f_c(x))$ where $\tilde{x}$ is the reconstructed approximation of the client's original data.

ii. Model Inversion:. Another key attack technique in data reconstruction is Model Inversion [27], applied in [10], which exploits the relationship between the gradients and the original input in a VanSL setting. In this approach, the honest-but-curious server iteratively optimizes both the original input x and the client model parameters θ to reconstruct the data. Instead of explicitly matching the gradients, the attacker performs coordinate gradient descent, where the optimization alternates between updating x and θ to minimize the mean squared error (MSE) for both input and parameter updates. The attack minimizes the following objective:

$$\tilde{x} = \arg\min_{\tilde{x}^*} MSE(\tilde{f}_c^*(\tilde{x}^*, \tilde{\theta}^*), f_c(x, \theta)) + \lambda \, \mathrm{TV}(\tilde{x}^*) \tag{15}$$

$$\tilde{\theta} = \arg\min_{\tilde{\theta}^*} MSE(\tilde{f}_c^*(\tilde{x}^*, \tilde{\theta}^*), f_c(x, \theta)) \tag{16}$$

Here, $\tilde{f}_c^*$ represents the random initialization of the client network, $\mathrm{TV}(\tilde{x}^*)$ is the Total Variation [71] to enforce smoothness, and λ is a regularization parameter. The attack is demonstrated in a single client single server (SCSL) setup and it can extend to multi-client configurations through two key properties: sequential client training and shared parameter updates. Since the server interacts with only one client at a time and all clients update the same parameter set, a multi-client setup functionally mirrors a single-client configuration with aggregated data.

iii. Functionality Stealing:. The functionality stealing attack in [92] allows a semi-honest server to train a *pseudo-client model* ($\tilde{f}_c$) that closely mimics f_c without knowing its structure in VanSL and USL setups. The server achieves this by using intermediate server models and a small set x_{pub} of size N. The goal is to make $\tilde{f}_c$ learn the mapping, making it functionally identical to f_c, while training the server model separately. Mathematically, the training objective for the pseudo-client model is to minimize the KL-Div (Kullback-Leibler Divergence), which measures the difference between the soft labels of $\tilde{z}_c$ and z_c rather than the hard labels. Once the functionality is stolen, the server trains a reverse mapping function f_s^{-1} to transform the feature space of smashed data back into the original input space. Since the pseudo-client model has already learned to produce feature representations similar to f_c, the server can fine-tune f_s^{-1} to improve reconstruction accuracy. The server can apply this attack to multi-client SL directly without any modification.

iv. Generative Adversarial Network (GAN):. A data reconstruction attack using GAN [52] is performed in a VanSL setup, where a semi-honest server trains a generator G to synthesize fake samples (x') and injects them into the learning process. By observing the responses of the honest participant, the adversary adjusts G to produce reconstructions that resemble the original training data. The adversarial objective is formulated as:

$$A_{DRA} = \min_G \max_D \frac{1}{|X|} \sum_{x \in X} \log D(x) + \frac{1}{|X|} \sum_{x' \in X'} \log(1 - D(G(x'))) \tag{17}$$

where D is the discriminator. Through an iterative adversarial process, the adversary refines G to generate highly accurate reconstructions. Similarly, Zeng et al. [91] demonstrate how a semi-honest server in a USL setup can reconstruct private client data using GANs. The attack begins with a shadow model, where the server approximates the client's model via an auxiliary dataset x_{aux}. A GAN discriminator, combined with cross-entropy loss, ensures the shadow model's output aligns with z_c. Once trained, an inverse model maps features back to the input space, optimizing data reconstruction with MSE loss. While this exploits server-side vulnerabilities, SL also faces threats from malicious clients in multi-client setups [91]. An honest but curious client can extract the server model via knowledge distillation, aligning an alternative model with the global server model by minimizing KL-Div loss. The client then performs feature space inversion, using GAN-based optimization to infer missing class samples. Instead of reconstructing data in pixel space, the attack operates in a low-dimensional noise space, improving efficiency and inference accuracy.

v. Feature Reconstruction. Feature-oriented reconstruction attack [87] exploits the representation preference encoded in the z_c that the client transmits to the server during training by a semi-honest server in the VansL setup. The attack follows a three-phase pipeline: First, the adversary constructs a substitute client $\tilde{f}_c$ by minimizing the distance between the client's smashed data and the generated features using a domain discriminator (DISC) network [20, 23], which distinguishes between features from different domains, and Multi-Kernel Maximum Mean Discrepancy (MK-MMD) [24, 49], a statistical measure to align distributions, to align feature spaces formulated as: $\min_{\tilde{f}_c} L_{DISC} + L_{MK-MMD}$ Second, an inverse mapping network, $\tilde{f_c^{-1}}$, is trained on public auxiliary data to reconstruct inputs from smashed data. Finally, during the attack phase, the trained inverse network is applied to the victim's smashed data snapshot z_c to reconstruct the private training data, given by: $\tilde{x} = f_c^{-1}(z_c)$.

Zhu et. al [96] achieve feature reconstruction by employing a simulator model trained on an auxiliary dataset that follows a similar distribution as the client's private data in a VanSL setup. This simulator aims to approximate the behavior of the client's private model, enabling the semi-honest server to infer private data without direct access. To enhance its effectiveness, adversarial regularization is introduced through a discriminator network D_1 that distinguishes real intermediate representations from synthetic ones, ensuring that the simulator learns indistinguishable representations. Once trained, a decoder model is used to reconstruct the original input features from the intermediate representations, with an additional discriminator D_2 guiding it to produce reconstructions that closely resemble real data.

C. Details of Label Inference Attacks

This section provides an in-depth examination of the label inference attacks discussed in Section 4.1.2, elaborating on their mathematical foundations and implementation details.

C.1. Details of Gradient-Based Label Inference:

The ExPLoit framework [37] formulates label inference as an optimization problem where the surrogate labels $\tilde{y}$ are iteratively refined. The server initializes random surrogate labels and computes $L' = H(\tilde{y}, \tilde{\hat{p}}y)$ where $(\tilde{p})$ are predictions from the surrogate model. Through backpropagation $\tilde{fc^*}$, the attack computes $\nabla z_s L'$ to minimize the ExPLoit loss function which consists of (1) gradient matching by minimizing $\mathbb{E}[\|\nabla_{z_s} L' - \nabla_{\tilde{z_s}} L\|^2]$, (2) label prior regularization via KL-Div $D_{\mathrm{KL}}(P_y \| P_{\hat{y}})$, and (3) cross-entropy regularization $\mathbb{E}[H(\tilde{y}, \tilde{p})/H(P_y)]$, ensuring label alignment. The final loss function is:

$$L_{\mathrm{ExPL}} = \mathbb{E}\left[\|\nabla_{z_s} L - \nabla_{\tilde{z_s}} L'\|^2\right] + \lambda_{\mathrm{ce}} \cdot \mathbb{E}\left[\frac{H(\tilde{y}, \tilde{p})}{H(P_y)}\right] + \lambda_p \cdot D_{\mathrm{KL}}(P_y \| P_{\tilde{y}}) \tag{18}$$

where λ_{ce} and λ_p are hyperparameters controlling the trade-off between objectives. By iteratively optimizing $\tilde{f}_c^*$ and the inferred labels, the attack successfully recovers private labels.

Erdogan et al. [10] attack methodology leverages gradient updates $\nabla_{z_c} L$ to achieve perfect label recovery accuracy by a semi-honest server. In their VanSL setup, the server receives z_c and $\nabla_{z_c} L$ from the client. The attack randomly initializes $\tilde{f}_c^*$ matching the client model's architecture and iteratively tests all possible labels $\tilde{y}$, selecting the one that minimizes gradient

differences:

$$\tilde{y}^* = \arg\min_{\tilde{y}} \text{MSE}\left(\frac{\partial L(f_c(f_s(x)), y)}{\partial \theta}, \frac{\partial L(\tilde{f}_c^*(f_s(x)), \tilde{y})}{\partial \tilde{\theta}}\right) \quad (19)$$

This optimization approach computes the Mean Squared Error between the gradients of the original model parameters (θ) and those of the surrogate model parameters ($\tilde{\theta}$). The attack succeeds due to the deterministic relationship between labels and gradient patterns.

The embedding swapping attack [4] operates through a structured process of gradient comparison. For implementation, the attacker first computes embeddings z_t and z_i for target and unknown samples respectively. During training, when x_i is used in a batch, the attacker first uploads z_i and records the corresponding back-propagated gradient ∇z_i from the server. In a later batch, the attacker swaps the embedding by sending z_t instead and observes the new gradient update $\tilde{\nabla} z_i$. The attacker determines whether x_i belongs to the target label based on the gradient norm ratio:

$$\frac{\|\tilde{\nabla} z_i\|2}{\|\nabla z_i\|2} \leq \alpha \quad (20)$$

and ensures that the gradient norm satisfies:

$$\|\nabla z_i\|_2 \leq \mu \quad (21)$$

where $\|\cdot\|_2$ represents the L2 norm, and α, μ are predefined threshold parameters. If both conditions hold, it suggests that swapping the embedding does not significantly impact the training loss, indicating that x_i likely belongs to the target label. This attack can be extended to multi-attacker cases.

Another attack proposed by Zhao et al. [93] enables a malicious client to infer private labels in VanSL through a structured three-step approach: dummy label initialization, auxiliary model training, and private label inference. The client first initializes an auxiliary model using semi-supervised clustering (K-Means) to generate structured dummy labels approximating the server's label distribution. A small set of labeled auxiliary samples serves as cluster centroids, improving training efficiency. The auxiliary model is then trained with a composite loss function, aligning its predictions with the server's outputs. Once trained, the model captures the server's decision boundaries, allowing the client to infer private labels by forwarding inputs through both models. The auxiliary model training process involves optimizing a composite loss function comprising three components:

- **Distance-based loss** (ℓ_d): Aligns gradients between the server and auxiliary model.
- **Performance-based loss** (ℓ_p): Ensures predictions match dummy labels.
- **Knowledge-based loss** (ℓ_k): Refines predictions using a small set of labeled auxiliary samples.

The iterative optimization process enables the auxiliary model to approximate the server's classification behavior without direct access to labels. Once trained, the auxiliary model allows the client to infer private labels by forwarding inputs through both models:

$$\tilde{y} \leftarrow \tilde{f}(z_c) \quad (22)$$

where $\tilde{y}$ represents the inferred labels, $\tilde{f}$ is the trained auxiliary model, and $f(X)$ is the client model's intermediate representation.

Xie et. al [86] propose a label inference attack against VanSL under a regression setting, addressing a gap in prior research focused on classification tasks against a semi-honest server. Their methodology consists of multiple stages. First, the attacker initializes dummy labels and a surrogate model that attempts to approximate the label model's behavior. The attack then employs a gradient matching strategy, where the attacker iteratively updates the dummy labels to minimize the distance between the gradients of the surrogate model and the gradients received from the label party. To enhance attack effectiveness, the authors introduce learning regularization: (1) Gradient Distance Loss, which minimizes the discrepancy between the gradients of the surrogate and the original model, (2) Training Accuracy Loss, which ensures that the surrogate model's predictions align with the dummy labels, and (3) Knowledge Learning Loss, which utilizes a small auxiliary dataset with known labels to guide the optimization process.

C.2. Details of Smashed-Based Label Inference:

After the completion of training, the server no longer receives gradients, making gradient-based attacks ineffective. However, during the inference phase, the adversary still receives z_c, which retains semantic similarities to x, thereby encoding label information and enabling label inference. Liu et al. [48] introduced three primary approaches for smashed data inference: Euclidean Distance-Based Matching, Clustering-Based Inference, and Transfer Learning-Based Inference. The Euclidean distance approach assigns z_c to the nearest stored reference sample z_{ref}, assuming they share the same label:

$$j = \arg\min_{ref} \|z_c - z_{ref}\|^2 \quad (23)$$

The clustering-based method further groups similar smashed data into clusters using K-Means clustering. Additionally, the Transfer Learning-Based Inference approach leverages pre-trained models to extract feature representations from the smashed data, improving label inference accuracy even when the cut layer is far from the output. Specifically, the adversary applies a pre-trained model $f_{\text{pretrained}}$ to transform z_c into a feature space where class separability is enhanced:

$$z_{\text{trans}} = f_{\text{pretrained}}(z_c) \quad (24)$$

Then, label inference is performed by matching z_{trans} to the closest reference embedding $z_{\text{ref,trans}}$ in the feature space:

$$j = \arg\min_{ref} |z_{\text{trans}} - z_{\text{ref, trans}}|^2 \quad (25)$$

By leveraging knowledge from pre-trained models, the adversary can generalize across datasets and improve inference robustness.

Extending beyond feature reconstruction, [96] also performs label inference in USL, where the client retains both the first and last layers of the model, keeping labels private. In this scenario, the server additionally trains a label simulator $\tilde{h}$ to approximate the client's final model layers. Instead of directly learning label mappings, $\tilde{h}$ processes the received z_{cr} and outputs $\hat{y}$. A key challenge in this setting is overfitting, if $\tilde{h}$ is trained solely on an auxiliary data x_{aux}, it may struggle to generalize to x. To address this, a random label-flipping mechanism is introduced, where a fraction of training labels are intentionally perturbed. This forces $\tilde{h}$ to learn generalized representations rather than memorizing specific label distributions from x_{aux}.

Huang et al. [30] proposed an attack on SFL that reconstructs pixel-wise accurate private training data from shared smashed data under a semi-honest server threat model. The attack comprises two phases: training and inference. During training, the server collects z_c from honest clients and generates pseudo-samples x' using a pseudo-sample generator trained with one-hot and entropy losses to ensure class-balanced generation. The model is then trained to learn the inverse mapping from smashed data to private images. In the inference phase, the model reconstructs private images from new smashed data. To enhance the effectiveness of the reconstruction, the model is optimized using cycle-consistency losses. The forward cycle consistency loss ensures that the reconstructed images, when processed again through the client-side model, produce outputs similar to the original smashed data. The backward cycle consistency loss ensures that processed smashed data can regenerate original-like images when passed through the model. The forward cycle-consistency loss is formally defined as:

$$L_f = \|\tilde{f}^{-1}(z_c) - x'\|^2 \tag{26}$$

These loss functions refine the inversion process, enabling more precise reconstructions of private data.

D. Details of Model Manipulation Attacks

This section provides an in-depth examination of the model manipulation attacks discussed in Section 4.1.4, elaborating on their mathematical foundations and implementation details.

D.1. Details of Adversarial Attacks:

The adversary in [14] perturbs feature representations by minimizing their cosine similarity. The perturbation is iteratively updated as:

$$\delta = \text{Clip}_\epsilon \left\{ \delta - \zeta \frac{\partial L_{\text{attack}}(z_s, z_s^*)}{\partial \delta} \right\} \tag{27}$$

where ζ is the update step size, ϵ is the perturbation budget, and $\text{Clip}_\epsilon\{\cdot\}$ ensures the perturbation remains within the allowed ϵ-ball.

The adversarial example in [26] $\tilde{x}$ is iteratively updated using a gradient-based method while ensuring that the perturbation δ remains within the predefined budget:

$$\|\delta\|_\infty \leq \epsilon \tag{28}$$

where δ is the adversarial perturbation applied to the input x, and ϵ defines the perturbation constraint.

D.2. Details of Backdoor and Poisoning Attack

i. Label Flipping Attacks: Kohankhaki et al. [43] investigated static label flipping attacks, where malicious clients systematically modify class labels within their local training data before transmission to the server. These modifications remain constant throughout the training process, causing the VanSL model to develop incorrect associations that degrade its performance through increased misclassification rates and reduced generalization capability. Their study examined poisoning scenarios by varying the number of malicious clients ($M \in \{0, 2, 4, 6, 8, 10\}$) and poisoning rate per client ($p \in \{0.25, 0.5, 0.75\}$). The attacks specifically targeted ECG readings, flipping labels between normal and abnormal classes, thereby introducing systematic errors in the model's ability to distinguish between normal and abnormal readings.

ii. Client-Side Backdoor Attacks: Yu et. al [89] introduce client-side backdoor attacks where a malicious client in the VanSL setup leverages its control over local training data to inject backdoor samples by modifying features or labels. If the client holds both the features and the labels, it directly backdoors the data by adding a trigger pattern and associating it with an incorrect label. If the server holds the labels, the attacker may use label inference techniques to identify and manipulate specific training samples. To enhance attack persistence, an auxiliary model is introduced to distinguish between the clean and backdoor samples in the feature space, improving the sensitivity of the model to backdoor patterns. This ensures that the attack remains effective without degrading the model's performance on the primary task, making detection challenging.

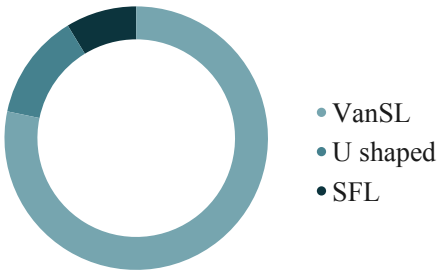

Figure 7: The distribution of split learning types in the surveyed literature on defense techniques.

Rieger et. al [69] also examine client-side backdoor attacks in the USL setup. The attack process begins with trigger insertion into a subset of its training data. The malicious client then trains its local model on both clean and poisoned data, ensuring that the backdoor is embedded while maintaining high overall model accuracy. Once the adversarial client completes training, it transmits the model update to the server, which forwards it to the next client. This gradual backdoor propagation allows the backdoor to persist through multiple training rounds, as subsequent benign clients unknowingly build upon the manipulated

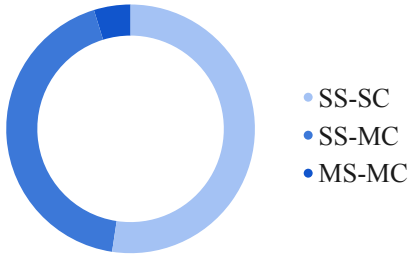

Figure 8: The distribution of client-server architectures in the surveyed literature on defense mechanisms.

model. Since SL does not reset model updates, the backdoor remains embedded even after multiple training rounds.

iii. Server-Side Backdoor Attacks: Further advancing the field of backdoor attacks, Tajalli et al. [74] investigated server-side backdoor attacks in VanSL, where an adversarial server injects backdoors without direct access to the client data. They proposed two strategies: Surrogate Client Attack and Injector Autoencoder Attack. In the Surrogate Client Attack, the server introduces a surrogate client $\hat{f}_c^*$ mimicking f_c and trains it on a poisoned dataset using a weighted loss function:$L_{\text{comb}} = \alpha L_c + (1-\alpha)\tilde{L}_c^*$ where α controls client influence in backpropagation. The Injector Autoencoder Attack trains an autoencoder on paired (clean, poisoned) smashed data and places it between the client's cut layer and the server's input pipeline to modify incoming activations in real-time. Their results suggest VanSL is resilient to backdoor attacks. Contrarily, Yu et al. [89] demonstrated that a malicious server can implant backdoors via feature space hijacking, aligning the client model's optimization process with a shadow model trained on a separate dataset. A discriminator network transfers the backdoor's effect to the client, while an auxiliary model maintains feature separability, ensuring stealth and preserving task accuracy. A discriminator network aids in transferring the backdoor's effect to the client model, ensuring it learns to associate trigger patterns with attacker-specified outputs. An auxiliary model further reinforces backdoor persistence by maintaining feature separability. Since this method does not alter raw client data, it remains stealthy while ensuring the primary task's accuracy is unaffected.

Recently, Yu et al. [90] introduced the SFI (Steal, Finetune, and Implant) attack framework, a sophisticated server-side backdoor attack in a VanSL framework, manipulating gradient updates without requiring access to raw client data. The framework operates in three distinct stages. In the Steal Stage, it constructs a shadow model $\tilde{f}_s$ using a limited shadow dataset x_s, labeling samples as either backdoor ($x_{sb} = 1$) or clean ($x_{sb} = 0$), while the shadow model learns from the server's main task model f_s. During the Finetune Stage, the framework optimizes the server model f_s, shadow model $\tilde{f}_s$, and auxiliary model f_a, where the auxiliary model assists in backdoor sample differentiation, ensuring stable backdoor encoding while maintaining main task accuracy. Finally, in the Implant Stage, the framework transfers backdoor capability to the client model by implementing a discriminator for adversarial training, forcing the client model to adapt feature encoding to match the shadow model while maintaining original training protocol integrity.

E. Supplementary Tables and Figures

We provide the general overview of attack and defense strategies and their features, such as implementation availability and SL style in Table 2. We also present the distribution of defense techniques across multi-client and multi-server settings in Figure 8 and the distribution of these SL types across the reviewed literature in Figure 7.

Paper	Split Learning Style	Implementation	Training Setup	Model Architecture	Attack Strategy	Defense Strategy
Fu et al.[18]	SFL	Available	SS-SC	DNN	NA	Monitoring - Gradient Detection
Erdogan et al.[13]	VanSL	Available	SS-SC	DNN (ResNet)	NA	Monitoring - Gradient Detection
SplitOut[11]	VanSL	Available	SS-SC	DNN (ResNet)	NA	Monitoring - Gradient Detection
Titcombe et al.[77]	VanSL	Available	SS-MC	DNN	NA	Data Perturbation
Mao et al. [52]	VanSL	No Open Source	SS-SC	DNN	NA	Architecture Modification - Data Decorrelation
Khan et al. [38]	VanSL	Available	MS-SC	DNN	NA	Secure Computation - Function Secret Sharing
Khowaja et al. [42]	VanSL	No Open Source	SS-SC	DNN	NA	Architecture Modification - Data Perturbation
ResSFL [44]	SFL	Available	SS-MC	DNN (VGG-11)	NA	Architecture Modification - Protocol Modification
Pham et al. [66]	VanSL	No Open Source	MS-MC	DNN	NA	Architecture Modification - Protocol Modification
PrivateMail [79]	VanSL	No Open Source	SS-SC	DNN	NA	Data Perturbation - Differential Privacy
Pham et al. [62]	VanSL	Available	SS-MC	DNN	NA	Architecture Modification & Differential Privacy
DISCO[73]	VanSL	Available	SS-SC	DNN	NA	Architecture Modification - Data Decorrelation
Turina et al. [78]	SFL	No Open Source	SS-MC	DNN	NA	Architecture Modification - Data Decorrelation
Gawron et al. [22]	VanSL	Available	SS-SC	DNN	NA	Differential Privacy
Khan et al. [41]	USL	Available	SS-SC	DNN (1D CNN)	NA	Secure Computation - Homomorphic Encryption
Khan et al. [39]	USL	Available	SS-SC	1D CNN	NA	Secure Computation - Homomorphic Encryption
Split HE[61]	VanSL	No Open Source	SS-SC	DNN	NA	Secure Computation - Homomorphic Encryption
Split Ways[40]	USL	No Open Source	SS-SC	1D CNN	NA	Secure Computation - Homomorphic Encryption
Nguyen et al. [57]	USL	Available	SS-SC	1D CNN	NA	Secure Computation - Homomorphic Encryption
Abuadbba et al.[1]	VanSL	Available	SS-MC	1D CNN	NA	Architecture Modification & Differential Privacy
SafeSplit[69]	USL	No Open Source	SS-MC	DNN	NA	Monitoring - Weight/anomaly detection
PSLF[83]	VanSL	No Open Source	SS-SC	DNN	NA	Architecture Modification - Data Decorrelation
CURE[35]	VanSL	Available	SS-SC	DNN	NA	Secure Computation - Homomorphic Encryption
Pasquini et al. [59]	VSL	Available	SS-MC	DNN	Data reconstruction - FSHA	NA
PCAT [92]	VanSL	No Open Source	SS-SC	CNN	Data reconstruction - Functionality Stealing	NA
Unsplit[10]	VanSL	Available	SS-SC	DNN	Data Reconstruction - Model Inversion	NA
Xu et al. [87]	VanSL	No Open Source	SS-SC	DNN	Data Reconstruction - Feature Reconstruction	NA
Li et al.[45]	VanSL	No Open Source	SS-SC	DNN	Label Inference - Gradient-Based Label Inference	NA
Liu et al.[48]	VanSL	Available	SS-SC	DNN	Label Inference - Gradient-Based Label Inference	NA
ExPLoit[37]	VanSL	No Open Source	SS-SC	DNN	Label Inference - Gradient-Based Label Inference	NA
VILLIAN[4]	VSL	No Open Source	SS-SC	DNN	Model Manipulation - Backdoor Attacks	NA
Yu et al. [90]	VanSL	No Open Source	SS-SC	DNN	Model Manipulation - Backdoor Attacks	NA
Tajalli et al.[74]	VanSL	No Open Source	SS-SC	DNN	Model Manipulation - Backdoor Attacks	NA
Fan et al.[14]	USL	No Open Source	SS-SC	DNN	Model Manipulation - Adversarial Attacks	NA
AdvUSL[26]	USL	No Open Source	SS-MC	DNN	Model Manipulation - Adversarial Attacks	NA
Kohankhaki et al.[43]	VanSL	Available	SS-MC	DNN	Model Manipulation - Poisoning Attacks	NA
Zhu et al.[96]	USL	No Open Source	SS-MC	DNN	Data Reconstruction- Feature Reconstruction	NA
Yu et al.[89]	VanSL	No Open Source	SS-MC	DNN	Model Manipulation - Backdoor Attacks	NA
SplitAum[93]	VanSL	No Open Source	SS-SC	DNN	Label Inference - Gradient Based Label Inference	NA
Zeng et al.[91]	USL	No Open Source	SS-SC	DNN	Data reconstruction - GAN	NA
Huang et al.[30]	SFL	No Open Source	MS-MC	DNN	Label Inference - Smashed Based Label Inference	NA
Gajbhiye et al.[19]	SFL	No Open Source	MS-MC	DNN	Model Manipulation - Poisoning Attack	NA
Xie et al.[86]	VanSL	Available	SS-SC	DNN	Label Inference - Gradient Based Label Inference	NA
Wu et al.[85]	SFL	Available	MS-MC	DNN	Model Manipulation - Poisoning Attack	NA
Ismail et al.[33]	SFL	No Open Source	MS-MC	DNN	Model Manipulation - Poisoning Attack	NA

Table 2: Comparison of prior Split Learning approaches, showing key attributes such as *VanSL* (Vanilla SL), *SFL* (Split Federated Learning), *USL* (U-Shaped SL), *VSL* (Vertical SL), *SS-SC* (Single-Server Single-Client), *SS-MC* (Single-Server Multi-Client), *MS-SC* (Multi-Server Single-Client), and *MS-MC* (Multi-Server Multi-Client). Model abbreviations include *DNN* (Dense Neural Network), *CNN* (Convolutional Neural Network) and *1D CNN* (1 Dimensional Convolutional Neural Network). The thick gray line separates *defense* papers (top) from *attack* papers (bottom) and NA stands for 'not applicable'.