Decoupling Vertical Federated Learning using Local Self-Supervision

Avi Amalanshu[*]
Indian Institute of Technology Kharagpur
Kharagpur, WB, India 721302
`avi.amalanshu@kgpian.iitkgp.ac.in`

Yash Sirvi
Indian Institute of Technology Kharagpur
Kharagpur, WB, India 721302
`yashsirvi@kgpian.iitkgp.ac.in`

David Inouye
Purdue University
West Lafayette, IN, USA 47906
`dinouye@purdue.edu`

Abstract

Vertical Federated Learning (VFL) enables collaborative learning between clients who have disjoint features of common entities. However, standard VFL lacks fault tolerance, with each participant and connection being a single point of failure. Prior attempts to induce fault tolerance in VFL focus on the scenario of "straggling clients", usually entailing that all messages eventually arrive or that there is an upper bound on the number of late messages. To handle the more general problem of arbitrary crashes, we propose Decoupled VFL (DVFL). To handle training with faults, DVFL decouples training between communication rounds using local unsupervised objectives. By further decoupling label supervision from aggregation, DVFL also enables redundant aggregators. As secondary benefits, DVFL can enhance data efficiency and security against gradient-based attacks. In this work, we implement DVFL for split neural networks with a self-supervised autoencoder loss. This performs comparably to VFL on a split-MNIST task and degrades more gracefully under faults than our best VFL-based method. We also discuss its gradient privacy and demonstrate its data efficiency.

1 Introduction

Federated Learning [19], or FL, was introduced by Google researchers as a strategy for distributed learning, addressing communication efficiency and data privacy. Distributed participants in FL training do not expose their data to any other party.

A variant is Vertical FL (VFL). Standard FL can be said to have a "horizontal" or "sample-parallel" division of data: each participant holds unique samples within a shared feature space. Conversely, VFL participants hold unique features of a common sample space, which is a "vertical", "feature-parallel" division. This allows guest agents who individually have incomplete and information about their target to learn meaningful joint representations without sharing data.

VFL has received nascent interest, with some real-world proposals and applications [16]. However, VFL requires careful synchronization which makes it difficult to engineer and scale. Training is contingent on the exchange of intermediate results between the guests and the host. A crashed connection or participant on the backward pass means a missed model update. On the forward pass, it is catastrophic: an unfulfilled model input.

[*]Corresponding author. Alternate email at avi.amalanshu@gmail.com and web at avi-amalanshu.github.io

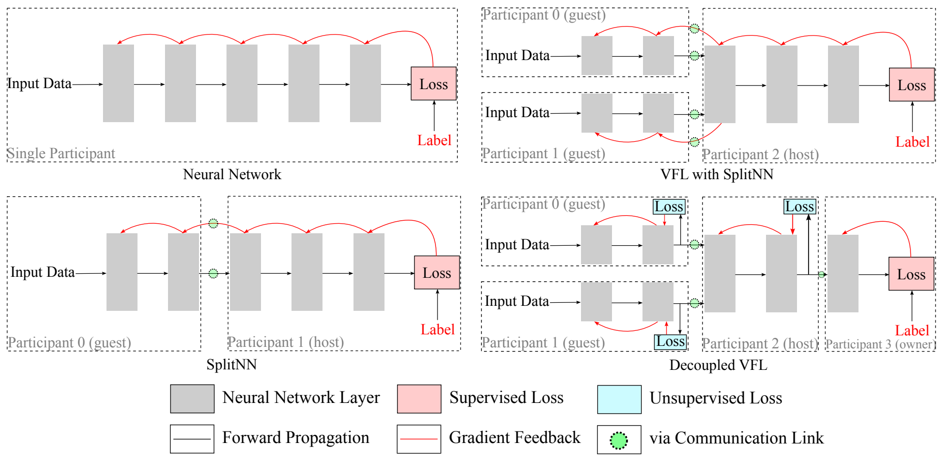

Figure 1: Diagram illustrating the distributed training of a neural network under SplitNN, VFL with SplitNN, and DVFL. The two input data for VFL and DVFL are partial features of the same entity.

We propose Decoupled Vertical Federated Learning (DVFL), where guests and hosts each train asynchronously on their own unsupervised objective before the label owner learns a transfer learning model from its labels. At an algorithmic level, DVFL avoids the forward and backward locking between hosts and guests. Hence, it is free of a single point of failure. Such “localized" training means that other participants can continue training even when one fails. Further, separating label inference from aggregation allows for redundant aggregators, further localizing faults.

The exchange of intermediate results is also the root of many other limitations of VFL including *inference attacks* (feedback received from the host has mutual information with labels and other guests' features) and the inability to train outside the intersection of all sample spaces. DVFL does not suffer from these problems.

We discuss these in more detail in the appendix (along with the notation and some implementation details). In the main paper, we present a basic justification for DVFL and experiments on fault tolerance and out-of-intersection learning.

2 Why Not a Systems-Level Solution?

The failure point can be partially mitigated within the VFL framework itself (with some extra bookkeeping). If the guest sends back sample indices and epoch numbers with each minibatch, the host can uniquely identify which request it belongs to. Now it may employ a timeout with a strategy to fill missing inputs; late messages may be identified and discarded. Alternately, the host may wait and poll unresponsive guests until they get at least one response for each request. So why DVFL?

Neither strategy handles faults on on backward passes, and *wait* is a busy-wait [13]. Further, there is no guarantee that training ever continues. In DVFL, live participants can continue training even when some fail. As for *timeout*, it is not trivial to choose a good replacement for missing inputs. We investigate some obvious options empirically.

3 Why Not a Straggler-Resilient Solution?

For VFL, there is limited work on crash faults. However, there is a line of work on the related case of straggling guests. The definition of the problem only extends to the count of failures being bounded as in FedVS [15], and/or late messages eventually arriving [22, 29, 11, 26]. Since a real system could have an arbitrary number and duration of crashes, these methods cannot generalize to crash faults. Typically, they do not address faults in the backward pass. Other paradigms (such as Gradient Assisted Learning [6]) which require periodic synchronization are similarly hamstrung.

Fully event-driven asynchronous VFL methods exist [3, 9], which are viable candidates to tackle crash faults. However, their convergence guarantees also assume all messages eventually arrive. Our experiments indicate that their performance with indefinite crashes may be graceful, but not as much as DVFL even at best. We discuss this in the appendix (§B).

Table 1: We compare DVFL to standard VFL. After filtering outliers by IQR, we report the mean and 2x the standard deviation of MNIST test performance when trained with various entities $i \in \{\text{connection}, \text{guest}, \text{host}\}$ susceptible to fail at a rate of $R_i^{(d)} = 0.3$. A failed resource has $R_i^{(u)} = \mathbb{P}_{\text{rejoin}} \in \{1, 0.5, 0.1\}$ to be available next time it is requested. SplitNN, the baseline VFL algorithm, cannot train at all when there are any faults. On the other hand, our DVFL implementation degrades gracefully with faults. Using an explicit fault handling strategy within VFL does allow it to train. But the degradation with faults is sharper than DVFL and the variance in model performance is higher. Under fault-free circumstances, SplitNN slightly outperforms our DVFL implementation. With minor adjustments, SplitNN can tolerate some faults, but our approach is more resilient and stable.

	Test Accuracy (%) with 0.3 Loss Rate									
Strategy	No Faults	Connection Loss			Guest Loss			Host Loss		
	$\mathbb{P}_{\text{rejoin}}$ = N/A	$\mathbb{P}_{\text{rejoin}} = 1$	$\mathbb{P}_{\text{rejoin}} = 0.5$	$\mathbb{P}_{\text{rejoin}} = 0.1$	$\mathbb{P}_{\text{rejoin}} = 1$	$\mathbb{P}_{\text{rejoin}} = 0.5$	$\mathbb{P}_{\text{rejoin}} = 0.1$	$\mathbb{P}_{\text{rejoin}} = 1$	$\mathbb{P}_{\text{rejoin}} = 0.5$	$\mathbb{P}_{\text{rejoin}} = 0.1$
DVFL-NN	97.80±**0.12**	**97.78±0.11**	**97.78±0.14**	**97.72±0.17**	**97.77±0.11**	**97.74±0.16**	**97.58±0.20**	**97.81±0.12**	**97.79±0.14**	**97.60±0.33**
VFL (SplitNN)	97.85±0.17	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VFL (SplitNN-skip)	**97.86**±0.16	97.50±0.54	96.60±0.51	77.21±5.34	97.44±0.57	96.65±0.45	80.22±4.31	97.65±0.41	97.59±0.50	97.29±0.89
VFL (SplitNN-zeros)	97.84±0.15	97.64±0.38	97.53±0.40	97.44±0.36	97.68±0.40	97.59±0.40	96.95±0.50	97.68±0.30	97.65±0.37	97.31±0.93
VFL (SplitNN-buffer)	97.85±0.17	97.66±0.38	96.85±0.74	93.45±2.10	97.59±0.58	96.76±0.70	91.78±1.70	**97.81**±0.32	97.71±0.40	97.69±0.51

4 DVFL with Self-Supervised Learning

DVFL splits the training into two phases coordinated by the owner. The first is an unsupervised and decoupled greedy *guest-host* representation learning phase. In this phase, the guest and host models learn feature extractors on their own unsupervised objectives. The second is a supervised *owner* transfer-learning phase where the label owner learns to aggregate encodings from the host space and produce label predictions. We do not consider faults in owner training. They do not affect the guests and hosts which train the bulk of the network. Further, even if there may be faults during owner training, the owner only needs to collect hosts' final encodings for each input– then, it may train its model offline whenever it is up.

Since guests do not rely on feedback from hosts to update their models, host or connection faults during the backward pass are irrelevant to the guest models. Hosts store activations from guests as they arrive and use the latest message for their forward pass, so a missing input on the forward pass is no longer critical.

The lack of end-to-end feedback combined with the three-tier hierarchy leads to significant robustness. The task of a host's model is only to learn an encoding of features in the joint guest domain, which is not an instance-specific task. Therefore, a host may still update its model meaningfully with some out-of-date activations. Similarly, guests may use their entire datasets as opposed to only members of the intersection. Besides leading to better models, using more data means a single fault does not affect the overall uptime as much. On the backward pass, faults only affect the crashed participant.

5 Experiments

As a toy example, we consider a system of $|\mathcal{G}|$ guests which can each see $\frac{1}{|\mathcal{G}|}^{th}$ of an handwritten digit from the MNIST dataset [5]. For our experiments, we set $|\mathcal{G}| = 4$ i.e. g_1's dataset contains the top 28×7 pixel patch of an MNIST image, g_2's dataset contains the second , g_3's contains the third, and g_4's contains the bottom 28×7 patch.

5.1 Baselines

DVFL is an alternate strategy to VFL. Naturally we compare DVFL for NNs to SplitNN, a direct application of VFL to NNs. Naive VFL implicitly employs the "wait" protocol. . We also modify the algorithm to implement the *timeout* strategy, imputing missing inputs with zeros (zeros), stale activations (buffer), and skipping that round altogether (skip).

For SplitNN hosts and DVFL owners, we use cross-entropy loss. For DVFL hosts and guests, We use a learnable decoder head and MSE reconstruction as a self-supervised objective.

5.2 Fault Tolerance

We use a fault model where a live resource has a fixed probability $R^{(d)}$ of dropping off at a given request (and $R^{(u)}$ vice-versa). How these faults are simulated are elaborated in the Appendix §D.2. For a given experiment, *which* resource fails (connection, guest, or host) is fixed. There are 4 DVFL hosts and 1 VFL host. These results are presented in Table 1.

Table 2: We exploit data outside the sample intersection to train DVFL's guests and hosts. For the owner model, we use labeled entities common to all datasets. This performs better than VFL, which can only use the common samples.

STRATEGY	TEST ACCURACY (%)			
	# OF LABELED SAMPLES			
	128	*256*	*512*	*1024*
DVFL	76.45	82.51	85.18	88.90
VFL (SplitNN)	65.07	77.51	83.09	87.26

5.3 Limited Intersection

We first fix a small labelled and aligned intersection size. The remaining sample space is shuffled and distributed evenly across all four guests. We try this experiment with 128, 256, 512 and 1024 labeled samples. We compare with SplitNN, which can only be trained on the labeled samples since the aggregator needs to calculate a supervised loss. Results are presented in Table 2

6 Discussion

6.1 Fault Tolerance

Model performance degrades with faults, but not catastrophically (unlike VFL with SplitNN). In ideal conditions, DVFL performance is slightly poorer than SplitNN's. This is not unexpected (see Limitations). Observe that the degradation with faults in DVFL is more graceful and *stable*.

In the case of SplitNN-based algorithms, a connection and guest fault are functionally identical. In the case of DVFL, there are some performance savings under connection faults– the corresponding guest and host can still meaningfully learn even in the absence of connection. Further, redundant hosts may continue training as normal.

6.2 Limited Intersection

Although VFL with SplitNN marginally outperforms our DVFL NN in the experiments with perfect conditions, DVFL significantly outperforms VFL on a limited labeled intersection. DVFL's advantage is greater with fewer labels (Table 2).

6.3 Gradient Privacy

VFL is performant while also completely eliminating the possibility of gradient-based inference attacks. Complex mechanisms can be developed on top of SplitNN to provide privacy guarantees within some bounds [12]. DVFL achieves complete privacy against gradient-based attacks without any extra computation and minimal performance loss.

6.4 Limitations

This study only considers an MNIST-based toy example. While this is a reasonable proxy for most benchmark VFL datasets (which are typically “easy", e.g. [18, 27, 21, 14]), it remains open to see whether this performance keeps up on “harder" datasets. Typically, local self-supervision does not scale well with deeper models (perhaps thanks to lesser mutual information with the task– analyzed by [24]). However, it is possible the information loss from connections between guests is a bigger bottleneck here. More investigation is needed.

7 Conclusions and Future Work

We present Decoupled Vertical Federated Learning, a strategy for joint learning on vertically partitioned data. Instead of adding complexity to cope with the data leakage and synchronization associated with BP, we eschew feedback altogether. Combined with the ability to reuse feedforward signals, this allows for fault tolerance and privacy in training. Training may continue even when there are faults, or there are a limited number of samples common to all guests.

We believe DVFL will enable large scale and democratic participation in joint learning tasks where VFL has been unable to provide solutions. e.g. street & private residential cameras predicting traffic incidents at a busy intersection. Participants might suffer from intermittent connection, might not all be able to see each training incident i.e. small $\mathcal{S}_{\text{guest}} \cap S_{\text{labels}}$, or might be concerned about privacy. DVFL is also more practical for existing cross-silo applications. Consider the typical example of various specialized healthcare providers predicting, say, cancer. Not only can the hospitals train asynchronously and not worry about faults, they need not worry about compromised hospitals launching gradient inference attacks. Further, they may now admit more hospitals to their system even if they have few patients in common.

Acknowledgments and Disclosure of Funding

A.A. and D.I. acknowledge support from NSF (IIS-2212097), ARL (W911NF-2020221), and ONR (N00014-23-C-1016).

References

[1] Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of CNNs. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 736–745. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/belilovsky20a.html`.

[2] Ceballos, I., Sharma, V., Mugica, E., Singh, A., Roman, A., Vepakomma, P., and Raskar, R. Splitnn-driven vertical partitioning, 2020.

[3] Chen, T., Jin, X., Sun, Y., and Yin, W. Vafl: a method of vertical asynchronous federated learning, 2020.

[4] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[5] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[6] Diao, E., Ding, J., and Tarokh, V. GAL: Gradient assisted learning for decentralized multi-organization collaborations. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=MT1GId7fJiP`.

[7] Erdoğan, E., Küpçü, A., and Ciçek, A. E. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, CCS '22. ACM, November 2022. doi: 10.1145/3559613.3563201. URL `http://dx.doi.org/10.1145/3559613.3563201`.

[8] Fu, C., Zhang, X., Ji, S., Chen, J., Wu, J., Guo, S., Zhou, J., Liu, A. X., and Wang, T. Label inference attacks against vertical federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1397–1414, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1. URL `https://www.usenix.org/conference/usenixsecurity22/presentation/fu-chong`.

[9] Gu, B., Xu, A., Huo, Z., Deng, C., and Huang, H. Privacy-preserving asynchronous federated learning algorithms for multi-party vertically collaborative learning, 2020. URL `https://arxiv.org/abs/2008.06233`.

[10] Gupta, O. and Raskar, R. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, August 2018. doi: 10.1016/j.jnca.2018.05.003. URL `https://doi.org/10.1016/j.jnca.2018.05.003`.

[11] Hu, Y., Niu, D., Yang, J., and Zhou, S. Fdml: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19. ACM, July 2019. doi: 10.1145/3292500.3330765. URL `http://dx.doi.org/10.1145/3292500.3330765`.

[12] Khan, A., ten Thij, M., and Wilbik, A. Vertical federated learning: A structured literature review, 2023.

[13] Lamport, L. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977. doi: 10.1109/TSE.1977.229904.

[14] Li, F.-F., Andreeto, M., Ranzato, M., and Perona, P. Caltech 101, Apr 2022.

[15] Li, S., Yao, D., and Liu, J. Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[16] Liu, Y., Kang, Y., Zou, T., Pu, Y., He, Y., Ye, X., Ouyang, Y., Zhang, Y.-Q., and Yang, Q. Vertical federated learning: Concepts, advances and challenges, 2023.

[17] Luo, X., Wu, Y., Xiao, X., and Ooi, B. C. Feature inference attack on model predictions in vertical federated learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, April 2021. doi: 10.1109/icde51399.2021.00023. URL `https://doi.org/10.1109/icde51399.2021.00023`.

[18] Markelle Kelly, Rachel Longjohn, K. N. The uci machine learning repository. URL `https://archive.ics.uci.edu`.

[19] McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Singh, A. and Zhu, J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 20–22 Apr 2017. URL `https://proceedings.mlr.press/v54/mcmahan17a.html`.

[20] Romanini, D., Hall, A. J., Papadopoulos, P., Titcombe, T., Ismail, A., Cebere, T., Sandmann, R., Roehm, R., and Hoeh, M. A. Pyvertical: A vertical federated learning framework for multi-headed splitnn, 2021. URL `https://arxiv.org/abs/2104.00489`.

[21] Sakar, C. O., Serbes, G., Gunduz, A., Tunç, H. C., Nizam, H., Sakar, B. E., Tutuncu, M., Aydin, T., Isenkul, M. E., and Apaydin, H. A comparative analysis of speech signal processing algorithms for parkinson's disease classification and the use of the tunable q-factor wavelet transform. *Appl. Soft Comput.*, 74:255–263, 2019. URL `https://api.semanticscholar.org/CorpusID:57374324`.

[22] Shi, H., Xu, Y., Jiang, Y., Yu, H., and Cui, L. Efficient asynchronous multi-participant vertical federated learning. *IEEE Transactions on Big Data*, pp. 1–12, 2022. ISSN 2372-2096. doi: 10.1109/tbdata.2022.3201729. URL `http://dx.doi.org/10.1109/TBDATA.2022.3201729`.

[23] Siddiqui, S. A., Krueger, D., LeCun, Y., and Deny, S. Blockwise self-supervised learning with barlow twins, 2023. URL `https://openreview.net/forum?id=uXeEBgzILe5`.

[24] Wang, Y., Ni, Z., Song, S., Yang, L., and Huang, G. Revisiting locally supervised learning: an alternative to end-to-end training. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=fAbkE6ant2`.

[25] Wu, Z., Li, Q., and He, B. Practical vertical federated learning with unsupervised representation learning. *IEEE Transactions on Big Data*, pp. 1–1, 2022. doi: 10.1109/tbdata.2022.3180117. URL `https://doi.org/10.1109/tbdata.2022.3180117`.

[26] Xia, W., Li, Y., Zhang, L., Wu, Z., and Yuan, X. Cascade vertical federated learning towards straggler mitigation and label privacy over distributed labels. *IEEE Transactions on Big Data*, pp. 1–14, 2023. doi: 10.1109/TBDATA.2022.3231277.

[27] Yeh, I.-C. and hui Lien, C. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2, Part 1):2473–2480, 2009. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2007.12.020. URL `https://www.sciencedirect.com/science/article/pii/S0957417407006719`.

[28] Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. Barlow twins: Self-supervised learning via redundancy reduction. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12310–12320. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/zbontar21a.html`.

[29] Zhang, Q., Gu, B., Deng, C., and Huang, H. Secure bilevel asynchronous vertical federated learning with backward updating, 2021. URL `https://arxiv.org/abs/2103.00958`.

A Background: Standard VFL

A.1 Notation

The rows of a dataset are features of a unique entity $\mathbf{x}$. We may serialize these entities using a sample index e.g. as $\mathbf{x}_j$. g_i is the i^{th} data-owning agent, member of set of guests $\mathcal{G}$. $\mathcal{F}_i$ is the set of features that guest g_i records. $\mathbf{x}_{j,\mathcal{F}_i}$ are the features $\mathcal{F}_i$ of the j^{th} entity. $m_i(\ \cdot\ ; \theta_i)$ is g_i's model, $\mathbb{R}^{|\mathcal{F}_i|} \to \mathbb{R}^{k_i}$ function parameterized by θ_i, which is learnable. $\mathcal{S}_i$ is the set of sample indices for which g_i has records. $\mathcal{S}_{\text{guests}} = \bigcap_{i=1}^{|\mathcal{G}|}$. h is the host agent, which owns labels. $\mathcal{S}_{\text{labels}}$ is the set of sample indices corresponding to labeled entities. $m_h(\ \cdot\ ; \theta_h)$ is the host's $\mathbb{R}^{k_1} \times \mathbb{R}^{k_2} \cdots \times \mathbb{R}^{k_{|\mathcal{G}|}} \to \mathbb{R}^{\text{out}}$ learnable model, which accepts the outputs of m_i and makes a task-relevant prediction.

A.2 The VFL Hierarchy

Guest: The guest role does not entail access to labels or other guests' data. The guest's private model m_i's task is to concisely represent the features $\mathcal{F}_i$ of $\mathbf{x}_j$ $\forall$ sample indices $j \in \mathcal{S}_{\text{guests}} \cap \mathcal{S}_{\text{labels}}$. Guests obtain these encodings and pass them to the host h.

Host: The host agent h executes every round of training. The host has access to target labels. The host passes encodings of all available features of $\mathbf{x}_j$ through its model and outputs a label prediction.

If the host has access to the full computational graph, it may adjust all parameters to better predict label $\mathbf{y}$, usually by minimizing the expected value of some loss function. Otherwise, it may update only its parameters θ_h and send gradients to guests, who may then update their own models.

Together, the guests and host solve the following joint optimization problem:

$$\theta^* =_\theta \mathbb{E}_{\mathbf{x}_j \sim X} \left[\ell(m_h \left(\hat{\mathbf{x}}_{j,1}, \ldots, \hat{\mathbf{x}}_{j,|\mathcal{G}|}); \theta_h \right), \mathbf{y}_j) \right] ,$$

where $\hat{\mathbf{x}}_{j,i} \equiv m_i(\mathbf{x}_{j,\mathcal{F}_i}; \theta_i) \in \mathbb{R}^{k_i}$.

A.3 Split Training for Neural Networks

SplitNN [10] is a faithful implementation of VFL for NNs. It was originally proposed as an algorithm to distribute the training of a global neural network across two agents by splitting the network depth-wise into two. At the "split layer", forward and backward signals are communicated between the two devices.[2] Ceballos et al. [2], Romanini et al. [20] extend SplitNN for vertically partitioned data by vertically splitting the shallower layers (see Figure 1). Effectively, the guest models m_i are NNs and host model m_h consists of a concatenation layer m_h^0 followed by an NN m_h^1.

A.4 Impracticalities in VFL Training

A.4.1 Single Points of Failure

In order to make a prediction on a batch of data, the host model $m_h : \mathbb{R}^{k_1} \times \cdots \times \mathbb{R}^{k_{|\mathcal{G}|}} \to \mathbb{R}^{\text{out}}$ requires inputs from all guest models. If an input is unavailable to the host model, no prediction can be made– a catastrophic fault. Similarly, local models' updates require gradients from the host, lacking which the model convergence slows down and may fail altogether. We consider the following fault model for a round of training:

[2]One might say that the standard SplitNN addresses a special case of VFL where the host has access only to labels and the (single) guest has access only to features. The original paper [10] also presents a protocol for training when the features are horizontally distributed across N guests.

- *Guest faults*: A guest may be unable to compute its output, update its model, or communicate.
- *Host faults*: The host itself may fail, and be unable to receive inputs or calculate gradients.
- *Connection faults*: The communication link between a guest and the host may be dropped during the forward or backward pass.

From a VFL host's perspective, a guest fault and connection fault manifest as a missing input to its model $m_h(\,\cdot\,;\theta_h)$ during training.

A.4.2 Data Intersection

Since VFL training is contingent on entity alignment, it is not usable in scenarios where the intersection of entities known to each guest is small. This is often the case when the number of guests is large. It could also be the case that, although $\mathcal{S}_{\text{guests}}$ is reasonably large, labels are not available for all its members, i.e., $\mathcal{S}_{\text{labels}} \cap \mathcal{S}_{\text{guests}} \subset \mathcal{S}_{\text{guests}}$.

A.4.3 Inference Attacks

Fu et al. [8], Luo et al. [17], Erdoğan et al. [7] show various methods curious guests may infer private information from gradients. Some attacks require training a generative model or somehow obtaining a small number of labels. Others are as simple as analyzing the sign of the gradient.

Typical defenses include differential privacy, gradient compression, homomorphic encryption and secret shares for gradients and model features. However, these methods usually have a negative effect on model convergence rate and tightness [12]. Moreover, implementing such cryptographic algorithms is computationally expensive.

B SplitNN-Zeros as a Skyline

We argue that under some constraints, SplitNN-zeros may be an upper bound on the performance of asynchronous and systems-level solutions.

Firstly, observe that when there are no skip connections, constrained solvers etc. SplitNN-zeros simply behaves as if missing inputs never existed to begin with– the gradient with respect to those inputs is zero and the output is only calculated based on the available information. From the perspective of asynchronous VFL, [3, 9] this is as if there is an oracle telling the host which clients will not respond to its next request– and then the host then only sending requests to live guests. Further, the host launches requests to all live guests simultaneously– so each model is updated based on all responses at once, reducing variance.

From the perspective of systems-level solutions (in particular, the *timeout* strategy), results on SplitNN-buffer show that there is no meaningful locality to be exploited. This suggests that unless we have a way of modelling missing inputs, SplitNN-zeros might be the best we can do. Trying to model inputs on the other hand would require prohibitive computation (perhaps a generative model). Moreover, an estimator would suffer from high variance, especially at the start of training. So we do not consider this alternative.

Finally note that in models where there *are* skip connections, constrained solvers etc. SplitNN-zeros will degrade in performance, since connections to dead inputs will get an error signal which should not exist.

C Computational Overhead

C.1 Periodic Communication

The owner initiates each round of training as well as communication for all agents. Since the training of guests and hosts are decoupled, the owner may ask guests to communicate their latent activations (and therefore start host training rounds) at arbitrary intervals. In VFL, guest model updates are dependent on host feedback. So there is every-shot communication: guests send every single output to the host.

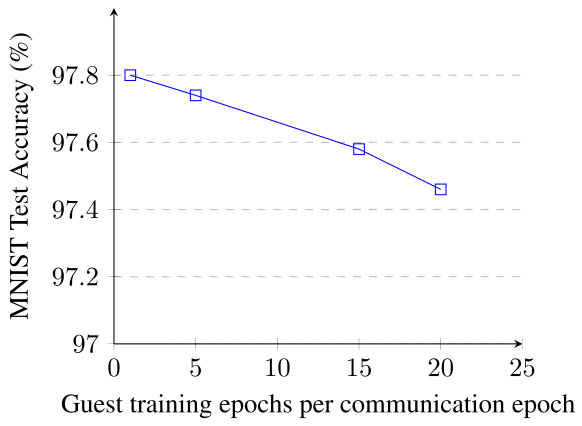

Figure 2: Model performance degrades with an increase in communication period, i.e. more communication is correlated with better performance.

With one-shot communication such as FedOnce [25], the volume of communication is greatly reduced, at the cost of model performance. Equivalently, Belilovsky et al. [1], Siddiqui et al. [23] demonstrate that updating all layers on local objectives on each forward pass performs better than fully training and freezing a layer before moving on to the next.

We study the tradeoff between communication cost and model performance by introducing a hyper-parameter K, which is the period (in guest epochs) at which *communication epochs* occur. During a communication epoch, every iteration of the guest model iteration is followed by a communication round i.e. guests write their activations to the communication registers, and hosts read them, concatenate them and store the concatenations in their input buffers.
If there is a host training iteration at every guest training iteration communication round, after the guests complete N epochs, the hosts will have

1. A model trained for $\lfloor \frac{N}{K} \rfloor$ epochs
2. The history of model inputs from each iteration in $\lfloor \frac{N}{K} \rfloor$ epochs of training

If the host must complete a certain number $M > \frac{|\mathcal{S}_{\text{guests}}|}{\text{batch size}} \times \lfloor \frac{N}{K} \rfloor$ iterations of training, it reuses these activations until the required number of iterations is met.

We simulate VFL training with 4 hosts and 0 fault rate. We measure test accuracy for the following communication periods:

- Every guest epoch ($K = 1$)
- Every 5 guest epochs ($K = 5$)
- Every 10 guest epochs ($K = 10$)
- Only once, after 20 guest epochs ($K = 20$).

Note that the dimension of the latent representations communicated by each guest to reach host is 80, and each dimension corresponds to a 32-bit floating point number. The MNIST training dataset has 60000 samples. If the total number of guest training epochs is N and the period of communication epochs is K, the total communication cost per guest is $|\mathcal{H}| \times \lfloor \frac{N}{K} \rfloor \times 60000 \times 80 \times 32$ bits. Each message received is saved by the host in the activation replay[3]. Consistent with results from Belilovsky et al. [1], Wu et al. [25], Siddiqui et al. [23], training the depth-wise split model parallely yields better model performance. We observe the tradeoff between communication cost and model performance. With 1.536GB, 0.307GB, 0.154GB and 0.077GB outgoing from each guest, the accuracies on the MNIST test set are 97.30%, 97.14% 97.08% and 96.84% respectively.

[3]Although this is not necessary if the host will not reuse activations, viz. a host wants to train for k iterations and it receives activations of k iterations from the guests. This is discussed in more detail in a future section.

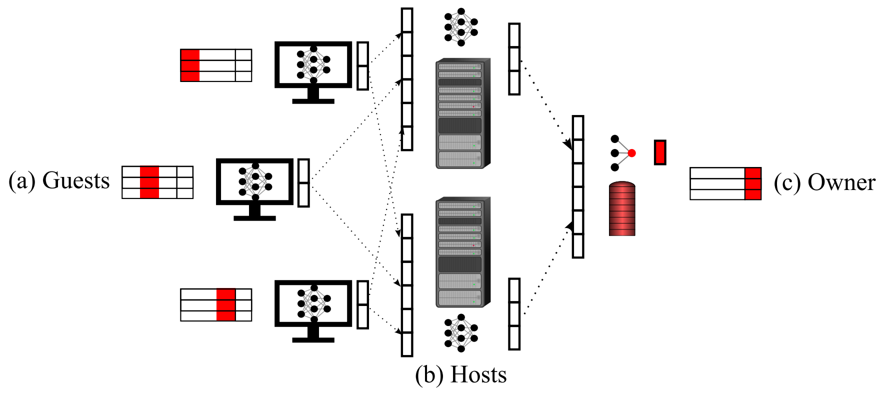

Figure 3: A DVFL system with three guests (a) and two hosts (b). Guests train their local models on unsupervised objectives and hosts also train their aggregating models on unsupervised objectives. After that, the label owner (c) trains a transfer learning model (such as a linear classifier head) on the encodings from the hosts.

C.2 Activation Replay Mechanism and Its Computational Overhead

A host gets its input from its activation replay buffer. That allows it to train asynchronously wrt the guests. The host can train independently of the guests as long as the guests have communicated earlier. Indeed, that is the case in our experiments– we simulate host models' training offline, after guest training. As discussed in Appendix C.1, this allows control over when the guests communicate.

Another way to look at this is: hosts do not need to train for the same number of iterations as guests communicate messages for. This includes the case where guests communicate every iteration, but hosts want to train more (or less) iterations. While we use 40 epochs to train the entire SplitNN model, we only use 20 epochs to train the DVFL guests. Guests need not remain online for any more time than required to train 20 epochs. So, not only does this allow for more flexibility, it also helps fault tolerance.

Computational Resources In order to maintain this asynchronicity, hosts need to maintain the activation replay buffer. For each message it receives, the size of the buffer grows by batch size $\times W_g$. Further, these expensive memory read/write operations also add time complexity to DVFL hosts' training. This overhead does not exist in standard VFL. This it is DVFL's primary drawback[4]. This extra computation can be mitigated to some extent since some models can be trained for fewer epochs as discussed above.

Also, it is important to note that it is not strictly necessary to save all activations. If a host trains for $\leq$ the number of epochs that guests train/communicate for, it does not need to reuse activations. In that case, a host model input can be deleted after it is used once.

In general, it is not necessary for any pair of participants (either belonging to any set: be it $\mathcal{G}$, $\mathcal{H}$ or $\{\text{owner}\}$) to train their model for the same number of iterations. This flexibility comes at the cost of space, but the upside from a system design and fault tolerance perspective is salient.

Table 3: Ablation study of model width (particularly, the dimensions of the outputs which are communicated) on model performance with DVFL. Our experiments in the main paper use $W_g = 320$ and $W_h = 160$. We run a hyperparameter sweep for each cell

W_h	**TEST ACCURACY (%)**			
	$W_g = 200$	$W_g = 240$	$W_g = 320$	$W_g = 400$
120	97.43	97.28	97.09	97.34
160	97.60	97.86	97.89	97.69
200	97.81	97.89	98.15	98.30

D Models, Hyperparameters and Implementation Details

D.1 Models

The “global", single-participant neural network (see Figure 3) being split is a shallow multi-layer perceptron. It is parameterized by W_g and W_h. W_g is the dimension of the concatenated output of all guests viz. the input to a host. W_h is the output dimension of a single host.

D.1.1 MNIST

Our batch sizes are 64.

The “guest" models have 2 layers, which squeeze the flattened $\frac{784}{|\mathcal{G}|}$-long input vector dimension to $\frac{400}{|\mathcal{G}|}$ via LeakyReLU, and then to $\frac{W_g}{|\mathcal{G}|}$ with ReLU activation.

The “host" models squeeze the W_g-dimensional input to $\frac{W_g+3W_h}{4}$ and then to W_h via LeakyReLU, with Leaky ReLU activation (with negative half-plane slope = 0.01).

The “owner" model squeezes the $W_h \times |\mathcal{H}|$ dimensional input to W_h and then to 40 with Leaky ReLU activations and finally to a 10-class prediction with an implicit softmax via the cross entropy loss function.

For DVFL experiments, the guest, host, and owner modules are separate objects simulated to reside on separate devices. In order to produce a reconstruction, they also contain decoder MLPs which undo the squeezing operation. The guest decoder is activated by a sigmoid function (since the input image is normalized to the range $[0, 1]$) and the host decoder is activated by a ReLU function (since the output of the guest model is also activated by a ReLU function and therefore cannot be negative).

For SplitNN experiments, the guest encoder architecture is identical to DVFL. The host model is a sequential module of the DVFL host encoder followed by the owner classifier.

We try DVFL for $W_g \in \{400, 320, 240, 200\}$ and $W_h \in \{200, 160, 120\}$. The classification accuracies are presented in Table 3. There two competing factors: the number of parameters (a monotonic function of $W_g \times W_h$) and the compression from host to guest. While more parameters implies a more generalizable (and hence hopefully accurate) model, an increase in W_g for a fixed W_h seems to suffer from some information loss, especially for larger W_h. It could be that the model eschews an encoding with more classification-relevant information when encodings with more reconstruction-relevant information are available thanks to the larger parameter space.

[4]The other source of overhead is the calculation of the self-supervised/unsupervised loss. For example, consider our experiments with MSE as the guest and host objectives. Along with our model, we need to train a decoder head that projects the guest output onto a space of the same dimension as input. Modern contrastive self-supervised losses e.g. Chen et al. [4], Zbontar et al. [28] typically also use a shallow projector network from the desired representation onto a latent space. The loss is calculated using the projected vectors. Intuitively, in the absence of a pre-computed ground truth label, the model must perform extra computation to generate the self-supervisory signals. However, the significance of this overhead depends heavily on implementation and is not strictly a bottleneck.

Table 4: Model dimensions used in our experiments on tabular datasets. The dimension of the input to a guest g_i's model is d_i. W_g is the dimension of the concatenated output of the guests' models. W_h is the dimension of the output of one host's model.

DATASET	W$_g$	W$_h$	ACTIVATION	
			GUEST	HOST
Credit Card	$\sum_i \lceil \frac{3d_i}{4} \rceil$	10	Leaky ReLU	Leaky ReLU
Parkinsons	$\sum_i \lfloor \frac{d_i}{4} \rfloor$	$\frac{\sum_i \lfloor \frac{d_i}{4} \rfloor}{4}$	Leaky ReLU	ReLU
CalTech-7	$256 \times \lvert\mathcal{G}\rvert$	256	Leaky ReLU	ReLU
Handwritten	$\sum_i \lceil \frac{3d_i}{4} \rceil$	$3\frac{\sum_i \lceil \frac{3d_i}{4} \rceil}{4}$	Leaky ReLU	ReLU

In the main paper, we use $W_g = 320$ and $W_h = 160$ for our experiments on MNIST. This choice enables good model performance while also demonstrating host models' ability to compress the data.

D.2 Fault Simulation

We simulate faults based on six hyperparameters:

1. **Connection Faults**: During each connection round, for each pair (g_j, h_i) we draw a sample from a uniform distribution with support $[0, 1]$. We track the current status of each connection using flags.
   - $R^{(d)}_{\text{connection}}$: If the connection between (g_j, h_i) was alive on the last iteration, we kill it if the sampled value is greater than $R^{(u)}_{\text{connection}}$.
   - $R^{(u)}_{\text{connection}}$: If the connection between (g_j, h_i) was dead on the last iteration, we revive it if the sampled value is lesser than $R^{(u)}_{\text{connection}}$.

   In DVFL, we only write g_j's activations to register $\mathbf{B}_{j,i}$ if. Similarly, for SplitNN, we check and update the flag before any forward pass from or backward pass to the split layer.
2. **Guest Faults**: Whenever a guest is called (i.e. there is a forward or backward pass), for each g_j we draw a sample from a uniform distribution with support $[0, 1]$.
   - $R^{(d)}_{\text{guest}}$: If the guest g_j was alive on the last call, we kill it if the sampled value is greater than $R^{(u)}_{\text{guest}}$.
   - $R^{(u)}_{\text{guest}}$: If the guest g_j was dead on the last call, we revive it if the sampled value is lesser than $R^{(u)}_{\text{guest}}$.

   In DVFL, we only update θ_j if the guest g_j is alive. If it is a connection epoch, we only write g_j's activations to $\mathbf{B}_{j,i} \forall i \in \{1, 2, \dots, |\mathcal{H}|\}$ if the guest is alive. Similarly, in SplitNN, we only allow a guest to send a forward pass and receive gradients if it is alive.
3. **Host Faults**: During each training iteration, for each h_i we draw a sample from a uniform distribution with support $[0, 1]$.
   - $R^{(d)}_{\text{host}}$: If the guest h_j was alive on the last call, we kill it if the sampled value is greater than $R^{(u)}_{\text{host}}$.
   - $R^{(u)}_{\text{host}}$: If the guest h_j was dead on the last call, we revive it if the sampled value is lesser than $R^{(u)}_{\text{host}}$.

   In DVFL, we only update $\theta_{H,i}$ if the host is alive. If it is a communication epoch, we also drop the last item in input buffer $\mathbf{A}_i$. In the case of SplitNN, we assume the host is able to call on the guests before failing. When a SplitNN host fails, it does not compute any gradients.

NeurIPS Paper Checklist

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our primary claims are that

   (a) DVFL performs comparably to VFL: Indeed, our implementation of DVFL using an autoencoder achieves $\approx 97.8\%$ on MNIST to $\approx 97.9\%$ for SplitNN. In the Appendix, we show DVFL's fault-free performance on benchmark VFL datasets (including tabular ones, which are typically difficult for neural networks) is also only marginally worse than VFL, while both VFL and DVFL outperform FedVS [15], a state-of-the-art fault tolerance and privacy solution for VFL.

   (b) DVFL can handle faults: Unlike standard SplitNN, training can continue under crash faults. The degradation of performance is graceful, moreso than if we were to implement a fault handling strategy directly with VFL.

   (c) DVFL is secure against gradient based attacks: Trivially, gradient-based inference attacks are not possible since there is no gradient feedback from other devices.

   (d) DVFL meaningfully exploits its full dataset: Performance on DVFL is better than VFL trained on only the intersection of datasets.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: It's Section 6.4

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The work is a system-level design and empirical results are presented. It is trivial that gradient inference attacks are impossible when there are no external gradients.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The algorithm used is described in pseudocode. The models, optimizers, hyperparameters are fully specified in the appendix. Our statistical methodology is described in the table captions.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [NA]

   Justification: N/A for the workshop version.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Available in Appendix §D.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our main claims are, in fact, bolstered by the standard deviations being lower for our method than the baseline.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: High-level discussion in §D

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Dedicated section that discusses both: §**??**

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks– our work promotes data privacy and security, as well as large-scale democratic participation in distributed learning.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the datasets used.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.