Vibe Hash: Proof of Useful Work by Verifiable Neural Network Training

Cole Heideman
vibehash.org

Abstract. Proof-of-work secures a ledger by making blocks expensive to produce, but the work itself is thrown away. We propose a network in which the work is the training of neural networks that people want trained. A worker commits to a Merkle tree over the checkpoints of a small, deterministic unit of training before a public random beacon chooses which segments staked checkers will replay. A disagreement is narrowed by a bisection game to one arithmetic operation, which the node recomputes on its CPU, so a verdict is correct if at least one checker is honest and online. Faking part of a unit is caught with probability at least equal to the part faked and costs the round's pay and half the cheater's stake, so cheating does not pay whenever honest work does. Every verified unit is paid in proportion to its floating-point operations: there is no lottery, and a laptop earns the same rate as the largest machine. The same protocol fine-tunes pretrained open models, including a 510 GB model trained by gaming PCs that each hold about 30 GB of it.

1. Introduction

Bitcoin [1] showed that a ledger can be kept without a trusted party by having nodes compete to find hashes below a target. The hashes have no use beyond the competition, and the competition is a lottery: each block's reward goes to one winner, which has pushed mining into pools and onto specialised hardware. Meanwhile, training machine-learning models has become one of the largest uses of computation in the world, while most capable processors, the graphics cards in gaming PCs, laptops and home servers, sit idle most of the day.

What is needed is a way to pay for training the way Bitcoin pays for hashing, without trusting the people who do it. The difficulty is verification. A hash is checked by computing one more hash; a step of training is checked only by doing it again. Work on proof of learning [3] has shown that a training run can be audited by replaying parts of it from saved checkpoints. In this paper we make that audit the core of a consensus protocol. Work is cut into units small enough for a slow machine to finish within a round. Every unit is committed to before anyone knows which part of it will be checked. The check is a replay assigned at random, a disagreement is settled by a game that ends in one operation any computer can judge, and the incentives make an honest unit worth more than a faked one. Because work is checked rather than raced, everyone whose work passes is paid in proportion to it. The same machinery fine-tunes pretrained open models, including one too large for any participant to hold.

2. Jobs and Work Units

A requester posts a job: a model specification, the hash of a dataset, training settings, a fee per round and a budget held in escrow. The network trains the job in rounds. Each round starts from the job's current weights W_r and divides the work into units of the form: starting from W_r, run S steps of stochastic gradient descent with batches chosen by the seed σ, saving a checkpoint every c steps. The batch of each step is derived from σ and the step number alone, so any part of a unit can be replayed on its own.

Units are deliberately small: the slowest eligible machine must finish one before the deadline. A faster machine receives more units, not larger ones, and a unit is never given to a machine without the memory to hold it. A worker's share grows with what it has delivered. A newcomer starts with a few units, may take at most twice as many in the next round if it delivered all of them, and drops back to what it delivered if it misses a deadline, so claiming more speed than one has gains nothing. At the end of the round, the accepted units are averaged into W_{r+1}, as in federated averaging [6]. The requester pays only for the share of each round that verified, and the trained weights are public: every worker receives them each round anyway.

3. Commitments

A unit with a checkpoint every c steps passes through checkpoints C_0, …, C_k, which divide it into k segments. The worker hashes each checkpoint, builds a Merkle tree [2] over the hashes, and before the compute deadline publishes a signed commitment

$$H(\text{ unit} \parallel \text{public key} \parallel \text{Merkle root} \parallel \text{salt }).$$

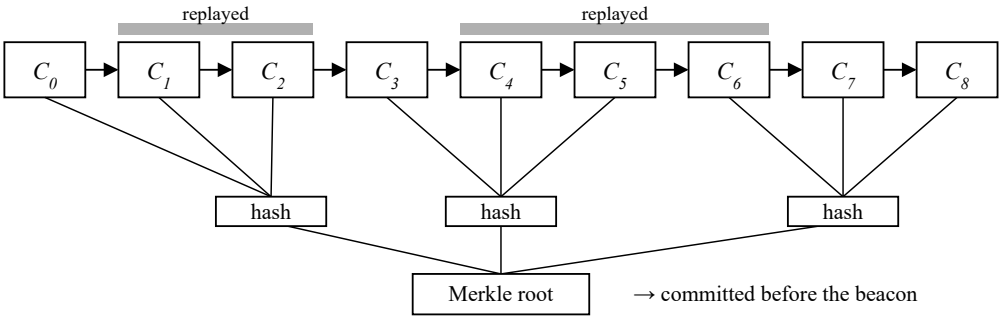


Figure 1: A unit's checkpoints, bound to one Merkle root before the beacon. Here the beacon then picks the shaded segments, $C_1 \to C_2$, $C_4 \to C_5$ and $C_5 \to C_6$, to replay.

The commitment reveals nothing about the result, so it cannot be copied, and it binds the worker to every checkpoint before anyone knows which will be examined. The first checkpoint must hash to the job's current weights, so a unit cannot start from invented ones. Workers upload their checkpoints to the round's node as content-addressed blobs before committing, which makes the reveal that follows a short message and counts the upload as part of the work.

4. The Beacon

After the compute deadline the round's randomness is

$$beacon = H(\text{ previous block} \parallel \text{round secret} \parallel \text{all commitments }).$$

The round secret is chosen by the validator leading the round when it opens it. The secret's hash is published at once, and the secret itself only in the sealed round's record. Without it, a worker who could see every commitment, for instance one alone in a round, could re-salt its own commitment until the beacon chose only segments it had computed honestly. Workers compare the published hash with the record, so the leader cannot change the secret after seeing the commitments either. From the beacon follow, for each unit, which q of its k segments are checked (three of eight by default, three of four for a fine-tune) and who checks them, and from the secret, where test submissions are planted.

5. Verification

Workers reveal the Merkle root, the salt, the final weights, and the checkpoints at both ends of every challenged segment with their Merkle proofs. A reveal that does not match its commitment is fraud. The checks, cheapest first:

- **Sanity.** The final weights must be finite, must equal the last checkpoint, and must differ from the start by a plausible amount. This stops garbage, and inflated updates meant to poison the model, without any replay.
- **Replay.** The unit's checkers, drawn at random among the staked workers able to run the job, never its holders, re-run each challenged segment from C_i and compare the result with C_{i+1}. It must lie within the job's tolerance, which absorbs rounding differences between processors, and within 1% of how far the segment actually moved the weights, which stops a worker from resubmitting untrained weights for a segment whose honest update is smaller than the tolerance, as happens late in training. A small model's claim has one checker and a fine-tune's two, so that one going quiet leaves the other; a claim neither answers is left unchecked, unpaid and out of the model.
- **Replication.** A random fraction of units is secretly given to two workers. Their final weights must agree; if they don't, a full replay, or for a fine-tune a dispute (Section 6), decides who lied.
- **Canaries.** Known-bad tasks are slipped into checkers' queues at positions drawn from the round secret: for a small model, a real task with one segment's update rescaled by 15 to 25%; for a fine-tune, whose rescaled Adam updates a lazy checker could spot, another unit's honest training under this unit's identity and seed. A checker that approves one is slashed, so approving without replaying does not pay.

A rejection is a claim, not a verdict. For a small model the node replays the disputed segment itself, and a checker who rejected honest work is slashed for the false accusation. For a fine-

tune a CPU is about a hundred times too slow, and the node should not need the model at all, so the rejection becomes a dispute. Committing and then not revealing forfeits the worker's whole round, but it is not slashed, because it cannot be told apart from a machine that crashed or lost its connection. Section 13 shows it is still unprofitable.

6. Disputes

A dispute has a defender, the worker whose claim is contested, and a challenger: a checker that rejected it, or, when two replicas of a unit disagree, each replica against the other. They play a bisection game [8, 9] down a tree that lays the unit out as a program, from its segments and optimizer steps through each layer's forward and backward pass to single operations, such as one matrix product, one normalisation or one step of Adam. Workers train by running this same program, so the state between any two operations is well defined: a set of named tensors, each committed to by a Merkle tree with 256 numbers to a leaf.

Moves alternate. The defender splits the current range into at most 64 children and publishes the state at each boundary, and loses if they do not chain from the range's own start to its end. The challenger recomputes them from the defender's own states and picks one whose result differs. After about six splits the pick is a single operation, and the defender supplies, with Merkle proofs, the inputs it read and one leaf of its output. A party silent past its deadline loses. A game takes at most fourteen moves, in a dispute phase added to the round.

The node is the referee. For each of the leaf's 256 numbers it computes, in 64-bit arithmetic from the defender's own inputs, the exact value and a tolerance for honest 32-bit rounding. A worst-case bound [10] grows with the length of every sum and would leave room to steer results, so the tolerance follows how rounding error actually behaves across graphics cards and processors, calibrated on honest runs of each operation. The referee uses only operations that IEEE 754 rounds correctly, with its own exponential and logarithm, so every validator reaches the same verdict to the bit. The slices of the model's weights it needs are read by byte range from the job's pinned copy and checked against the manifest's hashes, never taken from either party.

No validator takes the round's leader at its word. Every move carries a signed receipt like the round's other messages, and a validator replaying a game recomputes its starting states from the workers' reveals and checks the recorded weights against the manifest, so a leader cannot drop a move, invent a starting state or substitute weights unnoticed. Before a checker rejects a claim, it plays the game against its own replay; it starts only disputes it would win, and a checker that finds nothing at the leaf concedes and loses only that round's pay.

The loser is slashed. If the defender loses, its claim fails and the challenger collects the bounty. If the challenger loses, the whole slash is burned, so nobody profits from baiting an honest checker into a dispute. A claim's verdict is therefore correct if at least one of its checkers is honest and online. An honest worker does not lose, because the numbers it computed lie well inside tolerances calibrated on honest hardware. An honest checker that rejects work which

skipped computation wins, because a faked or extrapolated segment cannot be made of operations that each pass. And a dishonest checker's approval does not outweigh an honest one's rejection.

7. Rounds and Consensus

Each round produces one block. The block's summary carries the hash of the round's record: the workers' signed registrations, commitments, reveals and verdicts, the moves and evidence of any dispute, the round's rules, and the outcome of each check. Every choice made in the round is drawn from public seeds by a deterministic generator: which units each worker gets, which are duplicated, who checks whom, where the canaries go, which job runs. Anyone can therefore run the round again from its record and confirm that the block's summary follows from the signed messages.

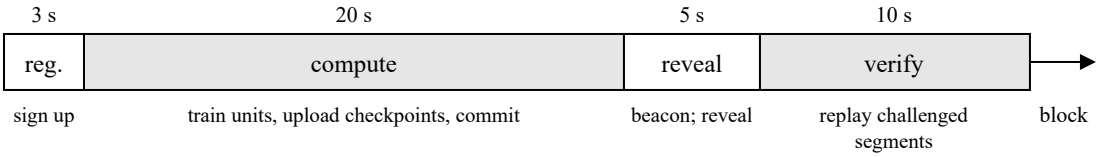


Figure 2: A round with the default phase lengths, drawn to scale. Jobs with larger models get longer phases, sized so a modest computer on a home connection can still move its checkpoints in time, and a fine-tune's round adds a dispute phase when a claim is disputed.

Validators hold stake and take turns leading rounds, in a stake-weighted order fixed by the block height alone. The leader runs the round with the workers and proposes the block together with its record. Every other validator applies the block and replays the record before voting, and voting follows Tendermint [4]. A block is final once validators holding more than two thirds of the stake have signed it, and those signatures are its commit certificate. A leader that pays itself extra, favours a worker or forges a verdict fails the replay, is voted down, and the next validator in the order leads instead. Rewards and slashes are never taken from the proposer: every node derives them from the round's summary and the chain's state. Useful work decides who is paid; stake decides who votes.

A node signs a receipt for every registration, commitment, reveal and verdict it accepts, naming the message's hash. A receipt for a message missing from its round's record is signed proof that the node dropped it.

8. Incentive

Each round has a pool, made of the block subsidy and the fee the job releases for the work that verified:

$$pool = subsidy(height) + \min(\, escrow, fee \times accepted\ units \,/\, units\ per\ round \,).$$

Eighty percent goes to workers in proportion to their verified training operations, fifteen percent to checkers in proportion to the operations they replayed, both weighted by reputation,

and five percent to the block's proposer. Anything unclaimed is burned rather than given to the proposer, so a proposer gains nothing by rejecting good work. Reputation is a multiplier between 0.5 and 1.0. It starts at 0.8, rises by 0.02 in each round with verified work, falls by 0.05 for a missed deadline, and drops to 0.5 on fraud. It moves per round rather than per unit, so a laptop builds it exactly as fast as a large machine.

Fraud costs half of the offender's stake, including coins unstaked in the last 20 blocks, which stay slashable, so a cheater cannot withdraw ahead of the verdict. Half of the slashed stake goes to whoever caught it and half is burned. The subsidy follows Bitcoin's shape stretched to Vibe Hash's block time: 3.125 VIBE a block, halving every 3,320,000 blocks, which is about four years at 38-second rounds, with at most 21,000,000 VIBE ever. Amounts are integers of 10^{-8} VIBE and rounding remainders are burned, so the supply is exactly the genesis allocation plus subsidies minus burns.

9. A Fair Start

Bitcoin needed no stake: a hash proves itself, so anyone could mine from the first block. Work that is only spot-checked needs a bond the cheater stands to lose, and if every stake has to be handed out in the genesis block, the first coins are a premine. Instead, anyone may register without stake, and pay to a worker below the minimum stake of 5 VIBE goes into its stake until it reaches the minimum. Until then the worker takes only units staked workers did not need, and never checks anyone else's work, so a swarm of free identities cannot approve each other's fakes. A fine-tune therefore assigns no work until two staked workers able to run it have registered, one to train a unit and one to check it. The first validators vote with bootstrap stake, which can never be withdrawn or spent and is burned if they leave, so it is outweighed as others earn stake and join. Every spendable coin starts from work.

10. Arithmetic Across Devices

Replays must agree across very different hardware. Every checkpoint is rounded to the job's precision before it is hashed, so a 32-bit job's Merkle leaves are the same whether its weights came from a graphics card or a CPU. Workers on graphics cards disable reduced-precision matrix modes and request deterministic kernels. Measured on a consumer NVIDIA graphics card against the reference CPU implementation, replays of segments of 5 to 500 steps on models of 69 thousand to 2.2 million weights differed by 4×10^{-9} to 8×10^{-8}, between 1,250 and 22,000 times inside the tolerance of 10^{-4}, and the same card gave identical bits on every run. The same measurements show why the update condition of Section 5 is needed: one honest 5-step segment of the 2.2-million-weight model moved the weights by only 4.3×10^{-5}, less than the tolerance.

Fine-tunes compute in 32-bit floats too, with the frozen 16-bit weights cast exactly inside each product. A four-step segment of a 0.6-billion-weight model trained on the graphics card and

replayed on the CPU differed by 2.6×10^{-6} of the segment's change, about 4,000 times inside the 1% rule.

11. Fine-tuning Pretrained Models

Much of the useful training today adapts a pretrained open model. A fine-tune pins an open-weight model at a commit, with a manifest of its tensors' hashes, and trains low-rank adapters [5]: each adapted projection computes $Wx + (\alpha/r)\cdot BAx$ with W frozen, and the adapters are the weights that units, checkpoints and averaging act on.

Averaging needs care: averaging A and B separately is not averaging the model, since the mean of B_iA_i is not the product of the means. We freeze A, drawn from the job's seed, and train only B [11, 12]; BA is linear in B, so averaging B is exactly averaging every weight's change. Each unit runs Adam from zero moments, and every checkpoint holds B with both moments, so any segment can be replayed alone, and checks compare B and each moment separately. The sanity check is Adam's own bound: no weight of B may move further than Adam can carry it whatever the gradients, 16.6 learning rates over a 16-step unit, so a boosted update fails without a replay.

The node never runs the model. Workers with graphics cards train, check and score, disputes are refereed one operation at a time, and the node checks hashes, Merkle proofs and Adam's bound and averages B. Three scorers measure each round's starting model on held-back data, and the record keeps their median loss for people to follow: nothing is paid or decided by it. In a first run on one computer, with the node still checking the work itself, three workers fine-tuned a 0.6-billion-weight Qwen3 model on 2,379 everyday conversations. Held-back loss fell from 2.461 to 1.119 in five rounds, and standard open-source tools measure the downloaded adapter at the same 1.1192.

At launch a fine-tune may use a model of up to 4 billion weights, such as Qwen3 0.6B and 4B, Llama 3.2 1B or SmolLM3 3B, with sequences of up to 2,048 tokens. That keeps every dispute's evidence and every round's record small enough for consensus; larger models and longer sequences follow.

12. Models Larger Than One Computer

A fine-tune as described needs the whole model on one card. The largest open models do not fit: DeepSeek-V4.1-Flash [13] has 763 billion weights in 40 layers, stored in 510 GB. A team could hold it as a pipeline [7], but on 20 Mbit/s uploads it would wait on its links 10 to 25 times longer than it computes. We take another route: train adapters only on the top K layers, four by default, and freeze the rest.

The frozen bottom's output for the job's text never changes, so it is computed once, as the job's preparation, and published under a Merkle root. Training the top layers from it is training the whole model with its bottom frozen, to the bit, as tests on a small model of the same architecture show. A miner holds the top four layers and the output head, 30.9 GB, with a shard

of the precomputed records, and trains units alone, exchanging only checkpoints. The trained layers compute in 32-bit floats: on the small model, replays on another device left *B* 8.9% apart in the model's own reduced precision and 0.0071% apart in 32-bit floats.

The preparation is mining work too. A rows unit streams a 6.8 GB range of the model's two n-gram tables (203 GB in all) and keeps the rows the job's text touches, each checkable against one ranged read of the pinned model. A layers unit runs a piece of four frozen layers, 29.6 GB, over one chunk of the text. Nobody needs anyone else online: each output waits in storage for the unit that reads it, checked against the digest its writer committed to, so the storage can withhold an object, which anyone holding its piece can recompute to the bit, but cannot alter one. Outputs are used as soon as they are committed; a unit a challenge overturns is redone with everything downstream of it, and the job pins its records' root once the last challenge window has closed.

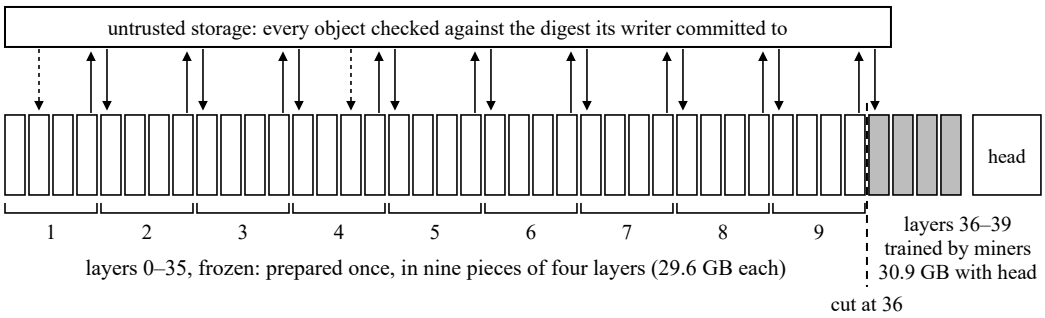


Figure 3: DeepSeek-V4.1-Flash's 40 layers and the cut at 36. The frozen bottom is prepared once, in nine pieces of four layers that any miner holding them can run, handing off through storage. Dotted arrows bring the rows of the model's two n-gram tables into layers 1 and 14. Miners train adapters on the top four layers (shaded).

Every unit is checked and judged like other work. A layers unit commits a 32-byte hash of each layer's output for each sequence. Anyone holding the piece can replay and challenge it, and the game bisects over its layers and then over one layer's operations, to one the node referees as in Section 6. The model's own reduced-precision arithmetic differs between devices by about 1% a layer, but that is drift accumulated over a layer, not any one operation's error: a checker replays each layer from the defender's committed input and rejects only when it differs by more than honest devices do, and the referee then judges single operations from the defender's own state, where drift cannot compound. Its FP8 and FP4 decoding and the model's choices of experts and keys are judged exactly. A training unit's checker replays a segment with the same four layers and only the records it used, proven against the job's root. A miner needs an NVIDIA card with 8 GB or more, 16 GB of RAM, since the experts are read from disk as their turn comes, and about 40 GB free on an SSD, of which about 30 GB is the model, downloaded once. Nobody holds the whole 510 GB.

The first job, "think shorter", teaches the model to reach correct answers with less reasoning, from grade-school math problems in GSM8K [14] with their short human-written solutions as

the reasoning. The pilot trains on 1,500 problems for three passes and holds back 500 test problems for the scorers. Its preparation splits the text into small chunks, so a piece fits an ordinary PC's memory.

13. Calculations

Suppose a worker skips the training of m of a unit's k segments and makes up their checkpoints. The beacon checks q distinct segments, unknown to the worker when it commits, so the worker escapes only if all q fall among the $k - m$ segments it trained:

$$P(\text{escape}) = \mathrm{C}(k - m, q) \,/\, \mathrm{C}(k, q) \;\le\; (1 - m/k)^{q}.$$

With the defaults $k = 8$ and $q = 3$:

m	*fraction skipped*	*P(caught)*
1	0.125	0.3750
2	0.250	0.6429
3	0.375	0.8214
4	0.500	0.9286
5	0.625	0.9821
6–8	≥ 0.750	1.0000

A fine-tune's unit has $k = 4$ and $q = 3$: skipping one segment is caught with probability 0.75 and skipping two or more always. In every case the probability of being caught is at least the fraction skipped. Replicated units and canaries only add to it. Now let u be the cost of computing one unit, R the worker's pay for the whole round and s its stake. Skipping m/k of a unit saves $(m/k)\cdot u$, and being caught loses $R + s/2$, so cheating pays only if

$$(m/k) \cdot u \;>\; P(\text{caught}) \cdot (\, R + s/2 \,).$$

Since $P(\text{caught}) \ge m/k$, that requires $u > R + s/2$: one unit would have to cost more than the pay for the whole round plus half the stake. If that were so, no one would mine honestly either. A worker that commits, sees which segments are challenged and withholds its reveal avoids the slash but loses R with the same probability, so backing out pays only if $u > R$, which is likewise a network nobody would mine on.

With two checkers, a fine-tune's bad claim is accepted only if one approves it and neither is an honest one that answers. If a fraction f of the eligible checkers would approve bad work and the honest ones answer, both of a claim's checkers are such with probability at most f^2: 0.01 at $f =$ 0.1, 0.25 at $f = 0.5$.

14. Limitations

- **One validator.** The public network has a single validator today; until others join, its operator could halt the network or refuse messages.

- **The hand-off storage.** The first large job's hand-offs pass through one storage bucket the operator pays for. It can withhold or delete data, which costs recomputation, but cannot alter it undetected.
- **Steering within the tolerance.** Tolerances must admit honest rounding with a margin, about seven times its typical size per sum, so a worker that does all the work could still nudge each result by a few parts in 100,000. It cannot skip work or fake results this way, and it gains nothing by doing so, but such nudges are not caught.
- **The frozen layers' arithmetic.** The preparation runs the frozen layers in the model's reduced precision, which differs between devices by about 1% a layer, so a unit's holder could bend each layer's output within the tolerance; a bitwise replay on the same class of hardware would close the gap.
- **Launch limits.** Fine-tunes start with models of up to 4 billion weights and sequences of up to 2,048 tokens.
- **Quality.** How well training only the top four of forty layers works is unmeasured until the first live job's held-back loss shows it, and lower loss alone does not show shorter reasoning at the same accuracy.
- **Status.** The software has not been audited. Vibe Hash is a test network, and VIBE has no cash value.

15. Conclusion

We have proposed a system that pays for the training of neural networks without trusting those who do it. Units are committed to before an unpredictable beacon chooses which segments are replayed, checkers repeat them within a tolerance that admits honest rounding but not skipped work, and a disagreement is narrowed to one operation any node can judge on a CPU. Cheating, lazy checking and false accusations each cost more than they gain, and all work that passes is paid in proportion. By preparing a model's frozen layers once, in pieces anyone can run, gaming PCs can fine-tune a model none of them holds. A test network implementing this design runs at vibehash.org; its coin VIBE has no cash value, and the software has not been audited.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] R. C. Merkle, "Protocols for public key cryptosystems," in Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122–133, April 1980.

[3] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran and N. Papernot, "Proof-of-Learning: Definitions and Practice," in IEEE Symposium on Security and Privacy, 2021.

[4] E. Buchman, J. Kwon and Z. Milosevic, "The latest gossip on BFT consensus," arXiv:1807.04938, 2018.

[5] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in International Conference on Learning Representations, 2022.

[6] H. B. McMahan, E. Moore, D. Ramage, S. Hampson and B. Agüera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Artificial Intelligence and Statistics, 2017.

[7] Y. Huang et al., "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," in Advances in Neural Information Processing Systems, 2019.

[8] R. Canetti, B. Riva and G. N. Rothblum, "Practical delegation of computation using multiple servers," in Proc. 18th ACM Conference on Computer and Communications Security, pages 445–454, 2011.

[9] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," 2017, arXiv:1908.04756.

[10] N. J. Higham, "Accuracy and Stability of Numerical Algorithms," 2nd edition, SIAM, 2002.

[11] L. Zhang, L. Zhang, S. Shi, X. Chu and B. Li, "LoRA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning," arXiv:2308.03303, 2023.

[12] Y. Sun, Z. Li, Y. Li and B. Ding, "Improving LoRA in Privacy-preserving Federated Learning," in International Conference on Learning Representations, 2024.

[13] DeepSeek-AI, "DeepSeek-V4.1-Flash," huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash, commit 314113f, 2026.

[14] K. Cobbe et al., "Training Verifiers to Solve Math Word Problems," arXiv:2110.14168, 2021.

Disclaimer: All product names, logos, and brands are property of their respective owners and are used purely for identification. Vibe Hash isn't affiliated with or endorsed by them.