GENOMEOCEAN ANYWHERE: PRIVATE WEBGPU INFERENCE FOR GENOME MOES

Anonymous authors
Paper under double-blind review

ABSTRACT

Genome foundation models are most useful where sequences are generated, yet the largest models need datacenter accelerators and a place to send private DNA. We ask whether a 15-billion-parameter genome mixture-of-experts (MoE) model can instead run on volunteers' web browsers, with the experts spread across many untrusted devices, without changing its predictions and without revealing the sequence to any single device. We build a system in which a trusted coordinator runs attention and routing while browser workers run every expert feed-forward network through hand-written WebGPU kernels, and we protect the expert inputs with real-valued Lagrange coded computing: each worker receives only a Gaussian-padded share, computes the expert's linear maps, and the coordinator decodes from any two of three workers. On GenomeOcean-MoE (8 experts, top-2 routing, 24 layers), the browser path matches native llama.cpp at every quantization level, the distributed path stays at the BF16 numerical noise floor (KL 0.0036 nats per token), and an unfitted latency model predicts decode time within 0.74% (median) under emulated wide-area links. We first show that plaintext expert inputs are not private: a probe recovers the token from a single vector at every depth, and one worker can identify the source genome from 300 unordered tokens with 92% accuracy. With coded experts, an adaptive attacker trained on shares falls to the most-frequent-token baseline, one worker's information about each token is bounded below one bit per forward pass, and the fidelity cost stays below the BF16 noise floor; in Chrome, coded decoding runs at 220 to 376 ms per token, depending on how much of the routing is hidden, and continues without replicas when a worker fails.

1 INTRODUCTION

Genome language models now score, annotate, and generate DNA at the scale of whole microbial genomes (Ji et al., 2021; Zhou et al., 2024; Dalla-Torre et al., 2025; Nguyen et al., 2024; Zhou et al., 2025b). The newest of them are sparse mixtures of experts (MoE): GenomeOcean-MoE upcycles a 4-billion-parameter dense model into eight experts per layer with top-2 routing, for 15.1 billion parameters in total (DOE Joint Genome Institute, 2026; Komatsuzaki et al., 2023; Jiang et al., 2024). Running such a model today means holding about 30 GB of weights on a datacenter accelerator and sending the sequence to whoever owns it. Both requirements are a poor fit for the people who produce sequences: field laboratories, clinics, and environmental surveys with modest hardware and data they may not share.

Two trends suggest an alternative. First, WebGPU gives every modern browser direct access to the local graphics processor, and browser engines already run dense language models of a few billion parameters (Ruan et al., 2024; Chen et al., 2025; ggml-org, 2025). Second, sparse experts are natural units of distribution: each token touches only two of eight experts per layer, and each expert is a self-contained feed-forward network whose weights are public. Volunteer computing (Anderson, 2020) and collaborative inference over the internet (Borzunov et al., 2023a;b) show that many donated devices can serve one model. Together they suggest a model that runs *anywhere*: a light coordinator keeps attention and routing, and the experts live in the browsers of whoever is willing to host them.

This design raises three questions that previous systems leave open. **Fidelity:** does running a genome MoE through browser kernels, quantized weights, and a network change its likelihoods,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

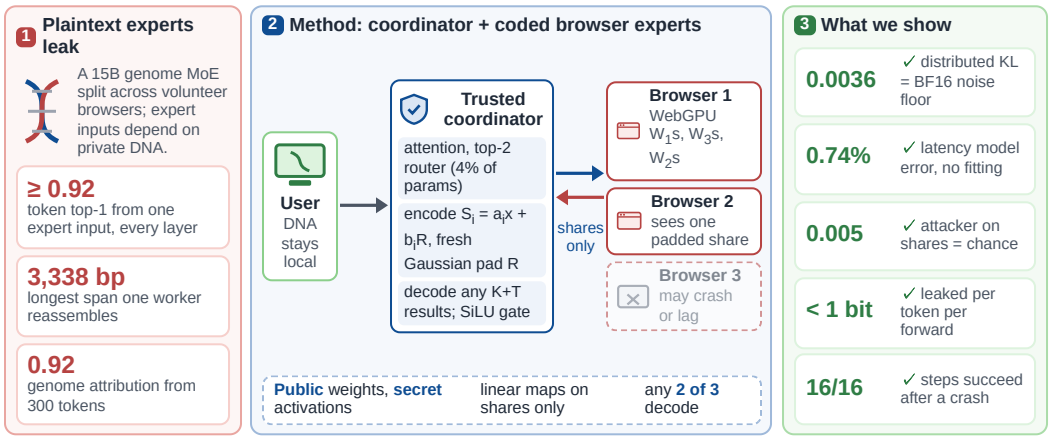

Figure 1: GenomeOcean Anywhere at a glance. (1) When a genome MoE is split across volunteer browsers, plaintext expert inputs reveal the sequence. (2) A trusted coordinator keeps attention and routing, and sends each browser worker only a Lagrange-coded share $S_i = a_i x + b_i R$ with a fresh Gaussian pad R; workers run the expert's three matrix products on shares with WebGPU kernels, and the coordinator decodes from any $K+T$ of N results and applies the SiLU gate. With $K=1$, $T=1$, $N=3$, any two workers suffice. (3) Main results, detailed in Section 5.

which is what genomic scoring depends on? **Performance:** what bounds speed when the experts sit behind wide-area links, and can the bound be predicted? **Privacy:** what does a volunteer learn from the expert inputs it receives, and can we stop it from learning the sequence? The last question is the hardest. It is tempting to believe that scattering tokens across many experts hides the sequence, but split-inference attacks on language models suggest otherwise (He et al., 2019; Pasquini et al., 2021; Qu et al., 2025; Dong et al., 2025; Thomas et al., 2025), and DNA fragments are exactly what genome assembly reconstructs.

We answer all three questions with a working system and measurements on GenomeOcean-MoE (Figure 1). Our contributions are:

1. **A browser-native expert-parallel runtime.** A coordinator runs embeddings, attention, and the top-2 router; headless Chrome workers run all 192 expert feed-forward networks through hand-written WGSL kernels that reach 19 to 20 TFLOP/s and match PyTorch to a relative error of 2.4×10^{-6} (Section 3).
2. **Fidelity at the numerical noise floor.** The browser path matches native llama.cpp at every GGUF quantization level, and the distributed path is indistinguishable from the difference between two BF16 implementations of the same model (KL 0.0036 nats per token; Section 5.1).
3. **A validated performance model.** An unfitted model with one round trip per layer predicts decode latency within 0.74% (median) over 48 emulated network conditions, and it correctly tells when a layer pipeline beats expert parallelism across sites (Section 5.2). Replication makes a worker crash invisible to the output (Section 5.3).
4. **Measured exposure and a coded defense.** Plaintext fragments leak the token at every depth and identify the source organism (Section 5.4). We adapt real-valued Lagrange coded computing (Yu et al., 2019; Soleymani et al., 2021) to expert feed-forward networks: workers see only Gaussian-padded shares, an adaptive attacker falls to chance, fidelity stays below the noise floor, and any two of three workers suffice (Section 5.5).

2 Related Work

Language models in the browser. TensorFlow.js brought neural network inference to JavaScript (Smilkov et al., 2019), and early measurements showed large gaps to native execution (Ma et al.,

2019). WebGPU closed much of that gap: WebLLM compiles models ahead of time with a machine-learning compiler (Ruan et al., 2024), WeInfer reduces WebGPU buffer and pipeline overheads (Chen et al., 2025), just-in-time kernel tuning improves edge browsers (Jia et al., 2024), and Transformers.js and llama.cpp now ship WebGPU back ends (Hugging Face, 2026; ggml-org, 2025). These systems run a whole model inside one tab. We instead split one model across many tabs and measure the fidelity of a genome model's likelihoods, not only throughput.

Decentralized and collaborative inference. Volunteer computing pools donated machines for science (Anderson, 2020). For neural networks, Learning@home trains decentralized mixtures of experts over the internet (Ryabinin & Gusev, 2020), and later work trains across heterogeneous or poorly connected devices (Diskin et al., 2021; Yuan et al., 2022; Ryabinin et al., 2023). Petals serves large models by assigning contiguous layers to volunteer servers (Borzunov et al., 2023a;b), and edge systems split transformers across nearby devices (Ye et al., 2024; Zhang et al., 2025). WDMoE places experts on wireless edge devices (Xue et al., 2024). Our system places every expert in a browser under exact top-2 routing, validates a latency model for this setting, and compares it against a layer pipeline measured on the same hardware.

Mixture-of-experts systems. Sparse expert layers scale model capacity at fixed compute (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022), and upcycling converts dense checkpoints into MoE models (Komatsuzaki et al., 2023). Serving systems optimize expert placement, communication, and kernels inside datacenters (Rajbhandari et al., 2022; He et al., 2022; Gale et al., 2023; Zhu et al., 2025), or offload experts on a single constrained device (Kamahori et al., 2025; Yi et al., 2025). MegaScale-Infer disaggregates attention and experts across datacenter nodes (Zhu et al., 2025); we apply the same split across browsers and untrusted parties.

Genome language models. DNA models range from masked encoders (Ji et al., 2021; Zhou et al., 2024; Dalla-Torre et al., 2025) to long-context and equivariant architectures (Nguyen et al., 2023; Schiff et al., 2024) and genome-scale generators (Nguyen et al., 2024); see Benegas et al. (2025) for a survey. GenomeOcean is trained on large-scale metagenomic assemblies (Zhou et al., 2025b), and its MoE variant is the largest openly released genome MoE we are aware of (DOE Joint Genome Institute, 2026). We study how such a model can be deployed outside the datacenter while keeping its likelihoods intact.

Privacy of distributed inference. Intermediate activations of split networks can be inverted to recover inputs (He et al., 2019; Pasquini et al., 2021), including prompts of collaborative LLM inference (Qu et al., 2025) and internal states of deep layers (Dong et al., 2025); permutation defenses leak as well (Thomas et al., 2025), and embeddings alone reveal much of their text (Morris et al., 2023). Split learning for health data rests on the same intuition that we test (Vepakomma et al., 2018). Cryptographic secure inference protects both inputs and weights with secret sharing or homomorphic encryption (Knott et al., 2021; Li et al., 2023; Gupta et al., 2024; Zhou et al., 2025a), and trusted hardware offers another route (Tramèr & Boneh, 2019); these protocols cost seconds to minutes per token. Coded computing takes a middle path: polynomial and Lagrange codes let a master outsource polynomial computations to untrusted workers with straggler tolerance and privacy against colluding sets (Yu et al., 2017; 2019; Lee et al., 2018; Shamir, 1979). Analog Lagrange coding extends this to real numbers with statistical privacy (Soleymani et al., 2021), and approximate coded inference handles nonlinear models without privacy (Jahani-Nezhad & Maddah-Ali, 2023; Soleymani et al., 2022); coded private training and single-layer edge inference use related ideas (So et al., 2021; Schlegel et al., 2022). We are, to our knowledge, the first to apply Lagrange coding to the experts of a transformer MoE, and to measure the resulting privacy against a learned attacker.

3 GenomeOcean Anywhere

Model. GenomeOcean-MoE is a Mixtral-style decoder (Jiang et al., 2024) with 24 layers, hidden size 3072, grouped-query attention, and in each layer 8 SwiGLU experts of width 8192 with top-2 routing; it has 15.13 billion parameters and a 4,097-token byte-pair vocabulary over nucleotides (DOE Joint Genome Institute, 2026; Zhou et al., 2025b). Each expert computes $y = W_2\,(\mathrm{SiLU}(W_1 x) \odot W_3 x)$ for an input row $x \in \mathbb{R}^{3072}$. The 192 experts hold 96% of the parameters.

Split of work. A coordinator holds the embeddings, attention, normalization, routers, and output head (0.63 billion parameters, 1.3 GB in BF16), together with the key-value cache. For each layer, it computes the router's top-2 choice per token, groups tokens by expert, and sends each group to the worker that hosts that expert. Browser workers are ordinary web pages that open a WebSocket to the coordinator, fetch their assigned expert weights once over HTTP, and then answer expert calls. The weights are public, so a worker learns nothing from them; what a worker must not learn is the expert input x, which is a function of the user's sequence.

WebGPU expert kernels. We implement the expert in WGSL, the WebGPU shading language, for BF16, Q8_0, and Q4_0 weights with in-register dequantization and FP32 accumulation. Small batches (at most 16 rows, the decoding case) use a matrix-vector kernel that streams each weight row once for all rows; larger batches use a 64×64 tiled kernel with shared-memory staging. Neither requires the optional `shader-f16` feature, which Chrome withholds on older NVIDIA drivers unless a Dawn toggle is set. Against PyTorch FP32 on the same inputs, the kernels have a maximum relative error of 2.4×10^{-6} and cosine similarity 1.0; at 1,024 rows one expert takes 7.7 to 8.1 ms on the GPU, or 19 to 20 TFLOP/s (Appendix B).

Single-browser mode. For users who can hold the full model on one device, we also run the unmodified llama.cpp engine (ggml-org, 2025), compiled to WebAssembly with 64-bit memory and its WebGPU back end, over GGUF files of the same model. This mode keeps the sequence on the user's device and serves as the fidelity and speed reference for browsers.

3.1 Coded experts

Plaintext expert inputs reveal the sequence (Section 5.4). We therefore never send x to a worker. Instead we use Lagrange coded computing (Yu et al., 2019), which lets a master evaluate a polynomial function on data held only in coded form by untrusted workers.

Encoding. Choose K data blocks, T pad blocks, and $N \geq K + T$ workers, with distinct interpolation points $\beta_1, \ldots, \beta_{K+T}$ and evaluation points $\alpha_1, \ldots, \alpha_N$ (Chebyshev nodes, for conditioning). With $B_j = X_j$ for $j \leq K$ and $B_{K+t} = R_t$ for random pads R_t, worker i receives

$$S_i \;=\; \sum_{j=1}^{K+T} \ell_j(\alpha_i)\, B_j, \qquad \ell_j(z) = \prod_{k \neq j} \frac{z - \beta_k}{\beta_j - \beta_k}. \tag{1}$$

For a linear map W, the values WS_i are evaluations of the degree-$(K+T-1)$ polynomial $Wu(z)$ at α_i, where u interpolates the B_j. Any $K+T$ results therefore determine $Wu(z)$, and the coordinator recovers $WX_j = Wu(\beta_j)$ by Lagrange interpolation. The other $N - K - T$ workers may crash or straggle.

Experts as two linear rounds. The SwiGLU nonlinearity is not a low-degree polynomial, so we code only the linear parts and keep the gate at the coordinator, which already runs attention in the clear. In round one, workers return $W_1 S_i$ and $W_3 S_i$; the coordinator decodes $W_1 x$ and $W_3 x$ and forms $h = \mathrm{SiLU}(W_1 x) \odot W_3 x$. In round two, it encodes h with a fresh pad and workers return $W_2 S'_i$, from which it decodes y. Workers run the same WebGPU kernels as before, without the fused activation.

Real-valued privacy. Over a finite field, LCC gives perfect privacy against any T colluding workers. Neural activations are real numbers, so we follow analog LCC (Soleymani et al., 2021) and draw pads from a Gaussian, $R_t \sim \mathcal{N}(0, \mathrm{diag}(\sigma r)^2)$, where r is the per-dimension root mean square of expert inputs on calibration data and σ is the pad scale. A share is then a Gaussian channel from the input. For a coalition C of at most T workers, with $A = E_{C,1:K}$ and $B = E_{C,K+1:K+T}$ the encoding coefficients, the leakage per share row is bounded by

$$I(X; S_C) \;\leq\; \tfrac{1}{2} \sum_d \log_2 \det\Big(I + K\, \frac{P_d}{(\sigma r_d)^2}\, (BB^\top)^{-1} AA^\top \Big), \tag{2}$$

where P_d is the second moment of dimension d of the input. We report this bound summed over every share a token produces in one forward pass (24 round-one shares and up to 48 round-two

shares), and we complement it with a learned attacker in Section 5.5. The price of a larger pad is precision: FP32 shares keep about $24 - \log_2 \sigma$ bits of the signal.

Hiding the routing. Which tokens reach which expert is itself informative (Section 5.4). We therefore support three modes: *routed* sends only the rows routed to each expert; *dense* evaluates round one for all eight experts on every token, so round-one requests are identical for all tokens; and *full* also evaluates round two for all experts, so no request shape depends on the routing. The coordinator discards the outputs of experts that were not selected.

4 Experimental setup

Data. We score held-out genomic sequence with the model's own likelihood, the quantity used for variant effect and sequence scoring. The evaluation set contains 48 windows of 1,024 tokens (47,535 tokens, about 5 bases per token) from *Escherichia coli*, *Pseudomonas aeruginosa*, and phage lambda, and a long set of 6 windows of about 8,000 tokens (47,604 tokens) from the same organisms. Probe attackers are trained on 600 windows (597,186 tokens) from three different organisms (*Bacillus subtilis*, *Streptomyces coelicolor*, and phage T4), so no training organism appears in evaluation. All windows come from RefSeq assemblies with a fixed random seed (Appendix A).

Reference and fidelity metrics. The reference is the released BF16 checkpoint run in PyTorch with FP32 router logits. For every test path we report the change in mean negative log-likelihood (dNLL, nats per token), the full-vocabulary KL divergence from the reference per token, and top-1 agreement, all on identical token ids. To calibrate what counts as a meaningful difference, we compare two correct BF16 implementations of the same model (stock Transformers against the FP32-router reference): they differ by KL 0.0036 nats per token and 95.7% top-1 agreement. We call this the *BF16 noise floor*.

Hardware and software. All components run in containers on a node with NVIDIA H100 80 GB GPUs. Browser workers are headless Chrome 154 with WebGPU on Vulkan; each worker is a separate browser instance with its own GPU process. Native baselines are llama.cpp with CUDA and with the Dawn WebGPU back end, built from the same source. Every comparison in a figure is measured in one session on the same GPU, with the compared configurations interleaved, and we report medians over repeated runs with interquartile ranges in the appendix.

Wide-area emulation. To study links between sites, the coordinator shapes every message with a one-way delay and a bandwidth cap per worker and direction, applied at the application layer. We measure 48 decode conditions (one-way delay 0 to 100 ms, bandwidth 10 Mbit/s to unlimited) and 45 prefill conditions, with shaped and unshaped runs interleaved so that slow drifts cancel.

Attackers. The exposure attacker is a linear or two-layer MLP probe that maps one expert input vector to the token at its position; it is trained on the disjoint organisms above and evaluated on held-out ones. For coded experts, the attacker knows the scheme, the encoding coefficients, and the pad scale, and retrains both probes on shares with fresh pads in every batch, using the share with the most favorable signal-to-pad ratio. The reference point for chance is the most frequent token (top-1 0.0051).

5 Results

5.1 The browser does not change the model

Figure 2a shows that the browser reproduces native likelihoods at every quantization level. For Q4_K_M, the NLL rises by 0.0212 nats per token in Chrome and by 0.0222 natively; for Q6_K by 0.0019 against 0.0024; for Q8_0 by 0.0008 against 0.0020. The ordering is the same in both runtimes, so the error budget is set by the choice of quantization, not by the browser. The same holds on 8,000-token windows, where the browser changes the Q4_K_M NLL by 0.0159 and native llama.cpp by 0.0175 (Table 3). Two independent browser runs produce bitwise identical outputs.

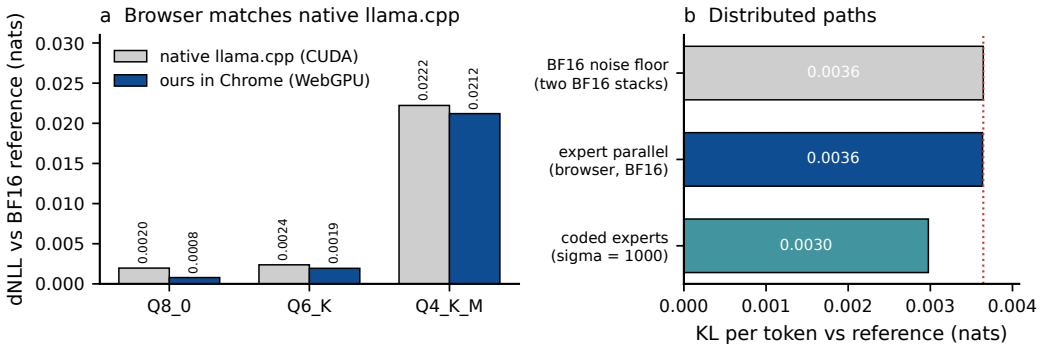

Figure 2: Fidelity on the 48-window evaluation set (47,487 scored tokens). (a) Change in mean NLL from the BF16 reference for native llama.cpp on CUDA and for the same GGUF files in Chrome. (b) KL per token from the reference. The dotted line is the BF16 noise floor, the difference between two correct BF16 implementations. Expert parallel: experts of layers 0, 11, and 23 in browser workers with BF16 weights. Coded experts: every expert of every layer computed on LCC shares with pad scale 1000.

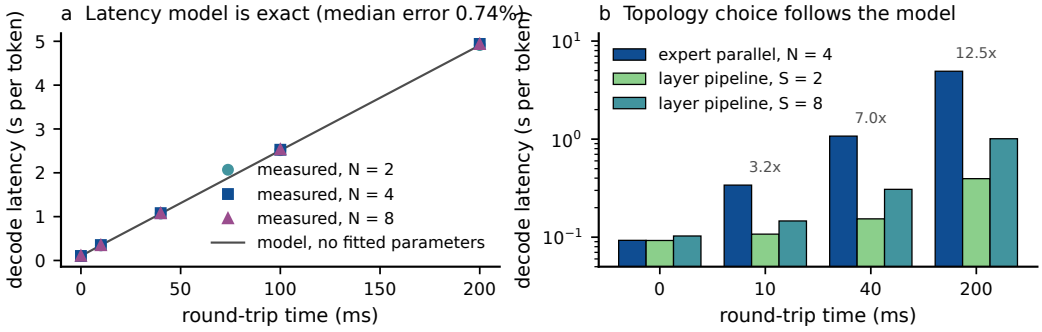

Figure 3: Decode latency under emulated wide-area links. (a) Expert parallelism with N browser workers, all 192 experts remote, unlimited bandwidth; markers are medians over 5 to 11 decode steps, the line is the unfitted model $T_0 + 24 \cdot \mathrm{RTT} + \mathrm{bytes}/\mathrm{bandwidth}$. (b) The same model predicts that a layer pipeline, which needs one hop per stage instead of one round trip per layer, decodes faster across sites; bars are measured on the same hardware in one session, with the speed-up of the two-stage pipeline over expert parallelism printed above each group.

The distributed path is just as faithful (Figure 2b). With the experts of three layers running in browser workers on BF16 weights, KL from the reference is 0.00364 nats per token, equal to the BF16 noise floor of 0.00364, and dNLL is -0.0007. Between 99.85% and 99.91% of the individual expert outputs returned by browsers are bitwise identical to PyTorch FP32 on the same inputs. When all 192 experts run remotely on Q4_0 weights, dNLL is $+0.0339$, matching a local Q4_0 control at $+0.0332$; quantization again accounts for the whole difference.

5.2 Performance is predictable, and the model picks the topology

On one H100, the expert-parallel runtime decodes at 104, 107, and 110 ms per token with 2, 4, and 8 browser workers and processes a 1,024-token prompt in 2.5 to 3.3 seconds (Table 5). Growing from 8 to 16 workers leaves prefill (1.63 against 1.61 s) and decode (111 against 112 ms) unchanged, because every token still touches two experts per layer. Each layer's exchange costs about 3.3 ms, and most of it is a fixed WebGPU submission and readback cost of about 2.3 ms; the GPU work itself takes 0.05 ms.

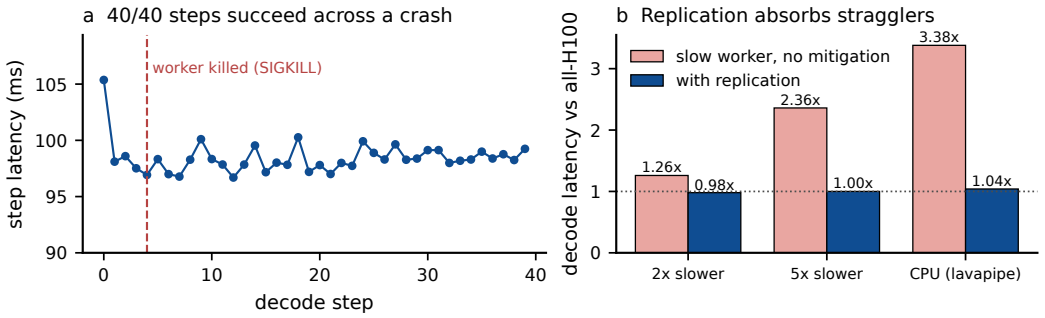

Figure 4: Robustness. (a) Eight browser workers with every expert on two workers (first response wins); one worker process is killed with SIGKILL at step 4. All 40 decode steps succeed, and the outputs of eight evaluation windows scored before and after the crash are bitwise identical. (b) Four workers, one of them slowed by a factor of 2 or 5, or replaced by a CPU WebGPU adapter (Mesa lavapipe); decode latency relative to four fast workers, without mitigation and with its experts replicated on a fast worker.

Across emulated links, a simple model explains decode latency: the unshaped time plus one round trip per layer plus bytes over bandwidth (Figure 3a). Without any fitted parameter, it predicts the 48 measured decode conditions with a median error of 0.74% and a maximum of 1.4%, and the 45 prefill conditions with a median error of 3.5%. A least-squares fit recovers a round-trip coefficient of 1.005 and an R^2 of 0.99999, confirming that the 24 sequential round trips are the whole story.

The model also tells operators which topology to use. A layer pipeline in the style of Petals (Borzunov et al., 2023a) needs one hop per stage rather than one round trip per layer. Measured with browser expert workers on the same hardware, a two-stage pipeline decodes at 0.154 s per token at 40 ms round-trip time, 7.0 times faster than expert parallelism at 1.07 s, and 12.5 times faster at 200 ms (Figure 3b). A hop model predicts all 33 shaped pipeline conditions within 1.3% for decode and 2.6% for prefill, and the pipeline's outputs are bitwise identical to the expert-parallel ones. Expert parallelism is therefore the right choice inside a low-latency site, where its prefill is two times faster (1.73 against 3.76 s), and a layer pipeline is the right choice between sites; both run the same browser expert workers.

5.3 Crashes and stragglers are absorbed

Volunteers leave without notice. With two copies of every expert, killing a worker in the middle of decoding has no visible effect (Figure 4a): all 40 steps succeed, the median step takes 98.3 ms, no step after the crash is slower than before it, and the coordinator detects the closed connection within 112 ms and serves the three pending requests from the replicas. Evaluation outputs scored before and after the crash are bitwise identical in NLL, log-probabilities, routing, and top-k. Replication adds only 2 to 3% to decode latency.

Replication also absorbs slow devices (Figure 4b). A worker that is 5 times slower raises decode latency by a factor of 2.36; with its experts replicated on a fast worker, the factor returns to 1.00. The same holds for a worker whose WebGPU adapter is a CPU rasterizer: 3.38 times slower without mitigation, 1.04 with replication, and the CPU adapter returns the same numerics as the H100 (99.88% of outputs bitwise equal to PyTorch).

5.4 Plaintext fragments are not private

Before designing a defense we measure the threat. A probe trained on other organisms recovers the token from a single plaintext expert input with top-1 accuracy of at least 0.92 at every layer (Figure 5a); the linear probe alone reaches 0.99996 at layer 0 and 0.57 at layer 12, and the routing decisions alone give 0.54. A worker holding one expert index across all layers receives 99.7% or more of all positions, reconstructs 46,766 of 47,535 tokens, and recovers correct contiguous spans

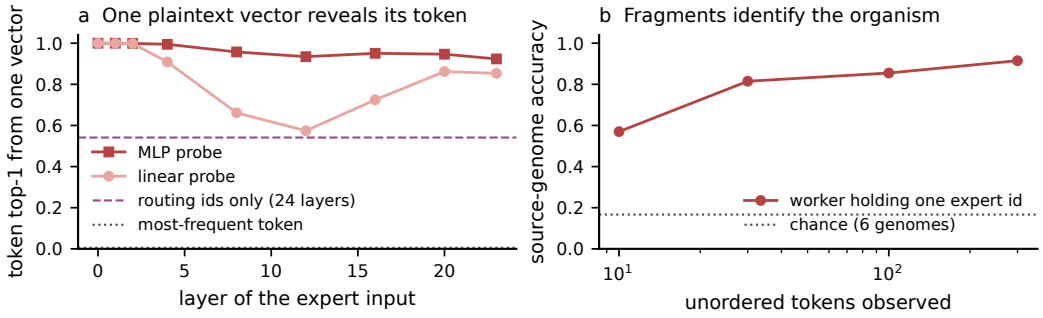

Figure 5: Exposure of plaintext expert inputs, measured on held-out organisms. (a) Top-1 accuracy of probes that predict the token from one expert input vector at a given layer; the dashed line uses only the routing decisions of all 24 layers. (b) A worker that hosts one expert index in every layer attributes the source genome among six candidates from a bag of unordered tokens (200 trials per point).

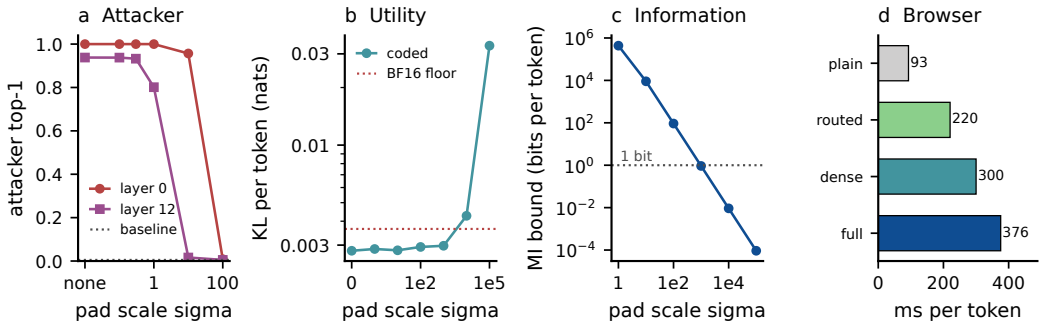

Figure 6: Coded experts with K=1, T=1, N=3. (a) An attacker that knows the scheme retrains linear and MLP probes on shares with fresh pads; best of the two, per layer. (b) KL per token between coded and uncoded forward passes over all 48 evaluation windows, with every expert of every layer coded. (c) Upper bound of Equation (2) on what one worker learns about a token, summed over all shares of one forward pass. (d) Decode latency in Chrome with three workers on one GPU at pad scale 1000, measured in one session: plaintext, coded with routed rows, coded with round one dense, and coded with both rounds dense.

of up to 3,338 base pairs. From as few as 30 unordered tokens it identifies the source genome among six candidates with 0.815 accuracy, and from 300 tokens with 0.92 (Figure 5b). Cheap mitigations do not help: Gaussian noise that already costs 0.18 nats per token leaves the retrained probe at 0.91 on layer 12; keeping the first eight layers local still leaves a layer at 0.96; shuffling token order has no effect on single-vector probes (Appendix D.1). Scattering the sequence across experts is therefore not a privacy mechanism.

5.5 Coded experts are private, faithful, and fault tolerant

Coding removes the leak. Against an adaptive attacker that retrains on shares, token accuracy falls to 0.016 on layer 12 at pad scale 10 and to the most-frequent-token baseline on both layers at pad scale 100 (0.0084 on layer 0 and 0.0051 on layer 12; Figure 6a). Layer 0 needs the larger pad because its inputs are close to the token embedding. The information bound confirms the attacker's failure from the other side: at pad scale 1000, one worker learns at most 0.92 bits about a token over an entire forward pass, less than a tenth of the 12 bits needed to name one token, and at pad scale 10^4 it learns at most 0.009 bits (Figure 6c).

Coding costs almost no accuracy. Over all 48 windows, with every expert of every layer coded, the KL between coded and uncoded outputs is 0.0030 nats per token at pad scale 1000 (Figure 6b), below the BF16 noise floor of 0.0036, and dNLL is $+0.0004$. The KL stays at 0.0028 to 0.0030 from no pad up to pad scale 1000 and only rises at 10^5, where FP32 has little precision left for the signal. The scheme generalizes: $K{=}2$ tokens per share gives KL 0.0032, and tolerating two colluding workers ($T{=}2$, $N{=}4$) gives KL 0.0033 (Table 10).

The defense runs in real browsers. With three Chrome workers at pad scale 1000, coded decoding takes 220 ms per token when only routed rows are sent, 300 ms when round one covers all eight experts, and 376 ms when no request depends on the routing, against 93 ms for plaintext in the same session (Figure 6d). Scored in the browser, the coded path agrees with the plaintext browser path to KL 0.0055 and dNLL -0.0014 over eight windows. Coding also provides fault tolerance for free: killing one of the three workers in the middle of decoding leaves all 16 steps successful, because any two results decode, and steps after the crash are no slower (285 to 303 ms against 305 to 318 ms before). What replication bought in Section 5.3 with a second copy of every expert, coding buys with one spare worker per group, while also hiding the input.

6 Discussion and conclusion

GenomeOcean Anywhere shows that a 15-billion-parameter genome MoE can be served from web browsers without changing its predictions. Browser kernels reproduce native likelihoods at every quantization level, distributed experts stay at the numerical noise floor of BF16 itself, and a one-line latency model predicts performance well enough to choose between expert parallelism and a layer pipeline before deployment. Replication hides crashes and slow devices, including devices whose WebGPU adapter is a CPU.

The central lesson concerns privacy. Distributing a sequence across experts feels private, but it is not: one plaintext expert input names its token, and a single worker can reassemble kilobase spans and identify the organism. Lagrange coded computing turns this around. Because expert weights are public and the coordinator already runs the nonlinear parts, each expert reduces to linear maps that can be evaluated on Gaussian-padded shares. At a pad scale of 1000, the learned attacker fails, the information bound drops below one bit per token per forward pass, the fidelity cost stays below the BF16 noise floor, and the same code that hides the input also tolerates a lost worker without replicas. Coded decoding in Chrome runs at 220 to 376 ms per token, well within the budget of genomic scoring, where the goal is to evaluate likelihoods of a given sequence rather than to chat.

Scope. Our guarantees follow the standard threat model of coded computing: honest-but-curious workers, at most T colluding workers per coding group, and a trusted coordinator, which can run on the user's own machine because it holds only 4% of the parameters. Privacy over real numbers is statistical, and we quantify it with both an information bound and a learned attacker; the full mode additionally removes routing from every request. All measurements use H100 GPUs, with wide-area links and slow devices emulated at the application layer, so that every number isolates one factor under controlled conditions.

Outlook. Two directions follow directly. Coded rounds could be overlapped across layers and batched across users to amortize their extra round trip, and the same coding applies unchanged to other open-weight MoE models. More broadly, public weights with secret activations is the common case for open foundation models; our results suggest that coded computing, long studied for matrix multiplication, is a practical privacy layer for decentralized inference with them.

AI use statement

We used generative AI tools for two purposes only. First, we used them to aid and polish the writing: large language models helped improve the phrasing, grammar, and readability of text drafted by the authors; they did not write the main content of the paper. Second, we used them to help generate synthetic datasets for our experiments, namely the Gaussian pads and the coded shares derived from them that are used in the coded-computing experiments (Sections 3.1 and 5.5 and Appendix C). The authors take full responsibility for the content of this paper.

ETHICS STATEMENT

This work uses only public genome assemblies of bacteria and bacteriophages from RefSeq and a publicly released model; it involves no human subjects and no human genetic data. Its purpose is to reduce the exposure of private sequences when models run on shared infrastructure. We report the exposure of plaintext expert inputs so that system designers do not rely on fragmentation for privacy, and we pair that analysis with a defense. The attack methods we describe are standard probing techniques and require access to intermediate activations that a correctly configured coded system does not reveal.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix A describe the data, reference, and metrics. Section 3 and Appendices B and C describe the runtime, the WebGPU kernels, and the coded protocol, including the encoding coefficients and the information bound. Appendix D reports every number shown in the figures, with sample sizes and interquartile ranges. The supplementary material contains the coded-computing implementation, the WGSL expert kernels, the figure script, the per-figure result files, and a CPU-only script that recomputes the headline numbers from them. The model is publicly available at the revision given in Appendix A.

REFERENCES

David P. Anderson. BOINC: A platform for volunteer computing. *Journal of Grid Computing*, 2020.

Gonzalo Benegas, Chengzhong Ye, Carlos Albors, Jianan Canal Li, and Yun S. Song. Genomic language models: opportunities and challenges. *Trends in Genetics*, 2025.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. In *Annual Meeting of the Association for Computational Linguistics*, 2023a.

Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin Raffel. Distributed inference and fine-tuning of large language models over the internet. In *Advances in Neural Information Processing Systems*, 2023b.

Zhiyang Chen, Yun Ma, Haiyang Shen, and Mugeng Liu. WeInfer: Unleashing the power of WebGPU on LLM inference in web browsers. In *Proceedings of the ACM on Web Conference 2025 (WWW '25)*, 2025.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Oteri, Christian Dallago, Evan Trop, Bernardo P. de Almeida, Hassan Sirelkhatim, Guillaume Richard, Marcin Skwark, Karim Beguir, Marie Lopez, and Thomas Pierrot. Nucleotide transformer: building and evaluating robust foundation models for human genomics. *Nature Methods*, 2025.

Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Quentin Lhoest, Anton Sinitsin, Dmitry Popov, Dmitry Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, Denis Mazur, Ilia Kobelev, Yacine Jernite, Thomas Wolf, and Gennady Pekhimenko. Distributed deep learning in open collaborations. In *Advances in Neural Information Processing Systems*, 2021.

DOE Joint Genome Institute. GenomeOcean MoE 8x2.4b, upcycled from GenomeOcean-4B. Hugging Face model card, DOEJGI/genomeocean-moe-8x2.4b-upcycled-from-4b, revision 8bfe18e8d5550851319b2d04dd7ef76a67d1db06, 2026.

Tian Dong, Yan Meng, Shaofeng Li, Guoxing Chen, Zhen Liu, and Haojin Zhu. Depth gives a false sense of privacy: LLM internal states inversion. In *USENIX Security Symposium*, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, 2023.

ggml-org. llama.cpp. `https://github.com/ggml-org/llama.cpp`, 2025.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2024.

Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. FasterMoE: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22)*, 2022.

Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.

Hugging Face. Transformers.js v4: Now available on NPM! `https://huggingface.co/blog/transformersjs-v4`, 2026.

Tayyebeh Jahani-Nezhad and Mohammad Ali Maddah-Ali. Berrut approximated coded computing: Straggler resistance beyond polynomial computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V. Davuluri. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics*, 2021.

Fucheng Jia, Shiqi Jiang, Ting Cao, Wei Cui, Tianrui Xia, Xu Cao, Yuanchun Li, Qipeng Wang, Deyu Zhang, Ju Ren, Yunxin Liu, Lili Qiu, and Mao Yang. Empowering in-browser deep learning inference on edge through just-in-time kernel optimization. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '24)*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations*, 2025.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure multi-party computation meets machine learning. In *Advances in Neural Information Processing Systems*, 2021.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*, 2023.

Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 2018.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. MPCFormer: Fast, performant and private transformer inference with MPC. In *International Conference on Learning Representations*, 2023.

Yun Ma, Dongwei Xiang, Shuyu Zheng, Deyu Tian, and Xuanzhe Liu. Moving deep learning into web browser: How far can we go? In *The World Wide Web Conference (WWW '19)*, 2019.

John X. Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M. Rush. Text embeddings reveal (almost) as much as text. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Michael Wornow, Callum Birch-Sykes, Stefano Massaroli, Aman Patel, Clayton Rabideau, Yoshua Bengio, Stefano Ermon, Christopher Ré, and Stephen Baccus. HyenaDNA: Long-range genomic sequence modeling at single nucleotide resolution. In *Advances in Neural Information Processing Systems*, 2023.

Eric Nguyen, Michael Poli, Matthew G. Durrant, Brian Kang, Dhruva Katrekar, David B. Li, Liam J. Bartie, Armin W. Thomas, Samuel H. King, Garyk Brixi, Jeremy Sullivan, Madelena Y. Ng, Ashley Lewis, Aaron Lou, Stefano Ermon, Stephen A. Baccus, Tina Hernandez-Boussard, Christopher Ré, Patrick D. Hsu, and Brian L. Hie. Sequence modeling and design from molecular to genome scale with Evo. *Science*, 2024.

Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

Wenjie Qu, Yuguang Zhou, Yongji Wu, Tingsong Xiao, Binhang Yuan, Yiming Li, and Jiaheng Zhang. Prompt inversion attack against collaborative inference of large language models. In *2025 IEEE Symposium on Security and Privacy*, 2025.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *International Conference on Machine Learning*, 2022.

Charlie F. Ruan, Yucheng Qin, Akaash R. Parthasarathy, Xun Zhou, Ruihang Lai, Hongyi Jin, Yixin Dong, Bohan Hou, Meng-Shiun Yu, Yiyan Zhai, Sudeep Agarwal, Hangrui Cao, Siyuan Feng, and Tianqi Chen. WebLLM: A high-performance in-browser LLM inference engine. *arXiv preprint arXiv:2412.15803*, 2024.

Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In *Advances in Neural Information Processing Systems*, 2020.

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. SWARM parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning*, 2023.

Yair Schiff, Chia Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. Caduceus: Bi-directional equivariant long-range DNA sequence modeling. In *International Conference on Machine Learning*, 2024.

Reent Schlegel, Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat. Privacy-preserving coded mobile edge computing for low-latency distributed inference. *IEEE Journal on Selected Areas in Communications*, 2022.

Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viégas, and Martin Wattenberg. TensorFlow.js: Machine learning for the web and beyond. *arXiv preprint arXiv:1901.05350*, 2019.

Jinhyun So, Basak Guler, and A. Salman Avestimehr. CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning. *IEEE Journal on Selected Areas in Information Theory*, 2021.

Mahdi Soleymani, Hessam Mahdavifar, and A. Salman Avestimehr. Analog Lagrange coded computing. *IEEE Journal on Selected Areas in Information Theory*, 2021.

Mahdi Soleymani, Ramy E. Ali, Hessam Mahdavifar, and A. Salman Avestimehr. ApproxIFER: A model-agnostic approach to resilient and robust prediction serving systems. In *AAAI Conference on Artificial Intelligence*, 2022.

Rahul Thomas, Louai Zahran, Erica Choi, Akilesh Potti, Micah Goldblum, and Arka Pal. Hidden no more: Attacking and defending private third-party LLM inference. In *International Conference on Machine Learning*, 2025.

Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019.

Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

Nan Xue, Yaping Sun, Zhiyong Chen, Meixia Tao, Xiaodong Xu, Liang Qian, Shuguang Cui, and Ping Zhang. WDMoE: Wireless distributed large language models with mixture of experts. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, 2024.

Shengyuan Ye, Jiangsu Du, Liekang Zeng, Wenzhong Ou, Xiaowen Chu, Yutong Lu, and Xu Chen. Galaxy: A resource-efficient collaborative edge AI system for in-situ transformer inference. In *IEEE Conference on Computer Communications (INFOCOM 2024)*, 2024.

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. EdgeMoE: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 2025.

Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems*, 2017.

Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, 2019.

Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. In *Advances in Neural Information Processing Systems*, 2022.

Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. EdgeShard: Efficient LLM inference via collaborative edge computing. *IEEE Internet of Things Journal*, 2025.

Yifan Zhou, Tianshi Xu, Jue Hong, Ye Wu, and Meng Li. CryptoMoE: Privacy-preserving and scalable mixture of experts inference via balanced expert routing. In *Advances in Neural Information Processing Systems*, 2025a.

Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. DNABERT-2: Efficient foundation model and benchmark for multi-species genome. In *International Conference on Learning Representations*, 2024.

Zhihan Zhou, Robert Riley, Satria Kautsar, Weimin Wu, Rob Egan, Steven Hofmeyr, Shira Goldhaber-Gordon, Mutian Yu, Harrison Ho, Fengchen Liu, Feng Chen, Rachael Morgan-Kiss, Lizhen Shi, Han Liu, and Zhong Wang. GenomeOcean: An efficient genome foundation model trained on large-scale metagenomic assemblies. *bioRxiv*, 2025b.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. MegaScale-Infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025.

A Data and reference

Model. We use `DOEJGI/genomeocean-moe-8x2.4b-upcycled-from-4b` at revision `8bfe18e8`, with the SHA-256 digest of each of its 15 weight shards checked against the release manifest (DOE Joint Genome Institute, 2026). The tokenizer is the released byte-pair tokenizer with 4,097 entries; sequences are encoded without special tokens.

Evaluation windows. Windows are sampled from RefSeq assemblies with seed 20260925 (Table 1). We tokenize each window and truncate to 1,024 tokens for the short set and to about 8,000 tokens for the long set. The attacker's training organisms are disjoint from the evaluation organisms, so probes cannot memorize evaluation sequences.

Table 1: Data sets. Tokens per window vary slightly because windows are cut in bases.

Set	Windows	Tokens	Bases per token	Organisms
Short evaluation	48	47,535	5.05	*E. coli*, *P. aeruginosa*, phage lambda
Long evaluation	6	47,604	5.04	same three
Attacker training	600	597,186	5.02	*B. subtilis*, *S. coelicolor*, phage T4

Reference and noise floor. The reference runs the BF16 checkpoint in PyTorch 2.7.1 and Transformers 4.53.3 with router logits computed in FP32. Its mean NLL on the short set is 4.3337 nats per token (perplexity 76.2). Stock Transformers in BF16 differs from it by dNLL -0.00008, KL 0.00364, and top-1 agreement 95.67%; router top-1 agreement is 99.16%. Running every operation in FP32 places both BF16 implementations at the same distance (KL 0.00363 and 0.00372), so neither is privileged. The NLL, KL, and top-1 metrics are computed on identical token ids; the scoring code refuses to compare rows whose ids differ.

B WebGPU expert kernels

Table 2 reports correctness and speed of one expert feed-forward network ($3072 \to 8192 \to 3072$) in Chrome on an H100. GPU time comes from WebGPU timestamp queries over back-to-back calls; end-to-end time includes upload, conversion, the two matrix passes, readback, and `mapAsync`. At one row, the GPU work takes 0.04 to 0.06 ms while a call takes about 2.3 ms end to end; this fixed per-call cost, not arithmetic, sets the decode latency of every browser runtime in this paper.

Table 2: Expert kernel correctness (maximum relative error against PyTorch FP32 on identical inputs) and speed. Medians over 7 repetitions.

Weights	Kernel	Rows	Rel. error	GPU ms	End-to-end ms	GPU TFLOP/s
BF16	matrix-vector	1	1.5×10^{-6}	0.057	2.3	2.7
BF16	matrix-vector	16	1.5×10^{-6}	0.287	2.5	8.4
BF16	tiled	1024	2.4×10^{-6}	8.09	22.1	19.1
Q8_0	matrix-vector	1	1.4×10^{-6}	0.040	2.4	3.8
Q8_0	tiled	1024	1.6×10^{-6}	7.72	21.0	20.0
Q4_0	matrix-vector	1	1.1×10^{-6}	0.041	2.3	3.7
Q4_0	matrix-vector	16	1.1×10^{-6}	0.284	2.4	8.5
Q4_0	tiled	1024	2.1×10^{-6}	7.84	20.6	19.7

Single-browser speed. Figure 7 compares the single-browser engine with the same engine compiled natively for Dawn WebGPU, at matched batch sizes (1,024 and 512) with flash attention. The browser sustains 71% to 76% of native WebGPU throughput for both prefill and decode at every quantization, so the sandbox itself costs about a quarter of native speed. Decode speed is flat across quantizations (59.7 to 64.8 tokens per second), although Q8_0 moves 1.7 times the bytes of Q4_K_M, which again points to a fixed per-dispatch cost.

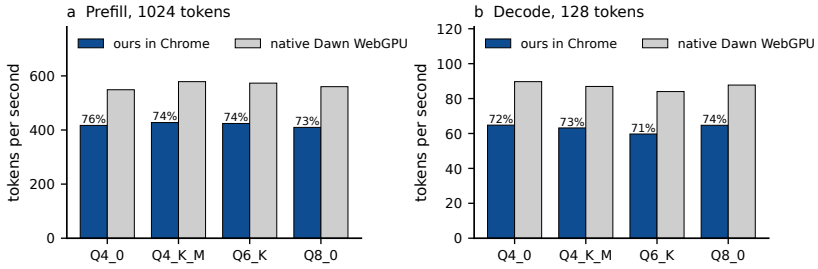

Figure 7: Single-browser throughput of GenomeOcean-MoE in Chrome against native Dawn WebGPU on the same GPU (medians over 5 repetitions; the percentage is browser over native). Both GPUs had no other process during the measurement.

C CODED COMPUTING DETAILS

Encoding coefficients. With K=1, T=1, N=3, the Chebyshev construction of Section 3.1 gives the encoding matrix E with rows (a_i, b_i) equal to $(-1.000, 2.000)$, $(0.618, 0.382)$, and $(-0.618, 1.618)$, so that $S_i = a_i x + b_i R$. Decoding from shares $\{0, 1\}$, $\{1, 2\}$, or $\{0, 2\}$ uses the inverse of the corresponding 2×2 block. Encoding and decoding run in FP64 at the coordinator, and shares travel in FP32.

Pad calibration. The pad of layer l has per-dimension standard deviation $\sigma r_{l,d}$, where $r_{l,d}$ is the root mean square of dimension d of the expert inputs of layer l on eight attacker-training windows; a separate r is computed for the round-two inputs h. Fresh pads are drawn for every layer, round, and call.

Self-test. For $(K, T, N) \in \{(1, 1, 2), (1, 1, 3), (2, 1, 3), (1, 2, 4), (2, 2, 5)\}$, a random linear map is recovered from every subset of $K + T$ shares at pad scale 1000 with a maximum relative error between 1.7×10^{-5} and 3.8×10^{-3}, the largest for the worst-conditioned subset.

Information bound. The bound of Equation (2) uses the held-out second moment P_d of each dimension, which exceeds the calibration value by a factor of up to 2.5 for x and up to 269 for a few dimensions of h; the bound therefore already accounts for inputs that are larger than the calibration data. We sum it over the 24 round-one shares and up to 48 round-two shares that one token produces in a forward pass; since pads are independent across shares, the sum bounds the total.

D ADDITIONAL RESULTS

Table 3: Fidelity against the BF16 reference. Short: 48 windows, 47,487 scored tokens, full-vocabulary KL. Long: 6 windows of about 8,000 tokens, 47,598 scored positions, KL over the top 64 renormalized tokens for both runtimes.

		Native llama.cpp (CUDA)			Ours in Chrome (WebGPU)		
Set	Weights	dNLL	KL	Top-1	dNLL	KL	Top-1
Short	Q8_0	+0.0020	0.0077	0.923	+0.0008	0.0065	0.926
Short	Q6_K	+0.0024	0.0111	0.907	+0.0019	0.0112	0.906
Short	Q4_K_M	+0.0222	0.0340	0.853	+0.0212	0.0335	0.855
Long	Q4_K_M	+0.0175	0.0306	0.852	+0.0159	0.0304	0.855

Table 4: Distributed fidelity on the short set. Expert parallel with experts of layers 0, 11, and 23 in four browser workers (BF16), and with all 192 experts in four workers (Q4_0) against a local Q4_0 control.

Path	dNLL	KL	Top-1
BF16 noise floor (two BF16 implementations)	−0.0001	0.0036	0.957
Expert parallel, 3 layers in browsers, BF16	−0.0007	0.0036	0.957
Local Q4_0 control, all experts	+0.0332	0.0519	0.824
Expert parallel, all experts in browsers, Q4_0	+0.0339	0.0519	0.824

Table 5: Expert-parallel speed on one H100 with all 192 Q4_0 experts in browser workers. Prefill of 1,024 tokens without cache (5 repetitions after warm-up) and decode (31 steps), medians with interquartile ranges. The 8 and 16 worker runs use round-robin placement and were measured in one session.

Workers	Prefill (s)	Decode (ms per token)
2	3.32 (2.92 to 3.32)	103.9 (103.2 to 105.7)
4	2.54 (2.51 to 2.58)	106.8 (104.6 to 108.5)
8	2.59 (2.43 to 2.63)	110.2 (109.1 to 112.2)
8 (round robin)	1.63 (1.59 to 1.68)	110.7 (108.9 to 115.6)
16 (round robin)	1.61 (1.60 to 1.62)	111.9 (110.5 to 114.4)

Table 6: Decode latency (s per token) under emulated links for expert parallelism with four workers, measured and predicted by the unfitted model, together with the measured two-stage and eight-stage layer pipelines from the same session.

Round trip (ms)	EP measured	EP model	Error	Pipeline S=2	Pipeline S=8
0	0.103	0.101	−1.9%	0.093	0.103
10	0.352	0.348	−1.0%	0.107	0.146
40	1.080	1.068	−1.1%	0.154	0.308
100	2.526	2.511	−0.6%		
200	4.940	4.909	−0.6%	0.395	1.009

Browser memory. In Chrome on this adapter, a single WebGPU buffer can hold up to 4 GiB minus 4 bytes, a tab allocated 14 GiB of GPU buffers without failure (the cap we set), and uploads from JavaScript reach 0.81 to 0.86 GB/s. With 64-bit WebAssembly memory the single-browser engine addresses 16 GiB, which is enough for every quantization of the model up to Q8_0 (a 15,316 MiB weight buffer on the GPU).

Routing statistics. Over 95,139 tokens, the eight experts of each layer receive between 0.87 and 1.25 times the uniform share of top-2 assignments, so no worker is starved or overloaded. A worker

Table 7: Change in mean NLL from the BF16 reference per organism of the short set. The browser follows native llama.cpp on every organism; the BF16 noise floor is shown for scale.

Weights	Runtime	*E. coli*	*P. aeruginosa*	Phage lambda
BF16	second implementation	-0.0008	$+0.0001$	$+0.0004$
Q8_0	native	$+0.0022$	$+0.0013$	$+0.0024$
Q8_0	browser	-0.0001	$+0.0010$	$+0.0014$
Q6_K	native	$+0.0045$	$+0.0021$	$+0.0006$
Q6_K	browser	$+0.0032$	$+0.0020$	$+0.0006$
Q4_K_M	native	$+0.0366$	$+0.0181$	$+0.0119$
Q4_K_M	browser	$+0.0349$	$+0.0191$	$+0.0097$

that hosts one expert index in every layer sees 99.72% to 99.96% of tokens, on average at about six of the 24 layers each. Consecutive tokens pick the same first expert with probability 0.35 on average against 0.125 by chance, and up to 0.72 in layers 8 to 10.

Prefill under limited bandwidth. For prefill, bandwidth rather than latency dominates, and more workers help because transfers proceed in parallel. At 100 Mbit/s per link, prefilling 1,024 tokens takes 29.1, 15.9, and 9.4 seconds with 2, 4, and 8 workers; the unfitted model predicts 29.5, 16.5, and 10.3 seconds. In the same session as Table 6, a two-stage layer pipeline at 100 Mbit/s prefills in 5.3 seconds, against 15.6 seconds for expert parallelism with four workers, because only one activation stream crosses each link.

Heterogeneous workers. In the straggler experiment, a slow worker sleeps for $(f-1)$ times its measured GPU time after each call. Without mitigation, a factor of 2 raises decode latency by 26% and prefill by 22%; with replication, decode returns to 0.98 times the baseline. At a factor of 5, speed-proportional re-placement brings prefill back to 1.08 times the baseline. A worker on Mesa lavapipe through ANGLE in WebGPU compatibility mode, pinned to eight CPU cores, returns results that are 99.88% bitwise identical to PyTorch, like the H100 workers, and with either mitigation the system runs at 1.00 to 1.26 times the all-GPU speed.

D.1 Exposure and mitigations

Table 8: Token top-1 accuracy of probes on single plaintext expert inputs, held-out organisms (47,535 tokens). The most frequent token gives 0.0051.

Layer	0	1	2	4	8	12	16	20	23
Linear	1.000	1.000	0.998	0.909	0.662	0.574	0.725	0.862	0.854
MLP	1.000	1.000	0.999	0.995	0.957	0.935	0.951	0.947	0.924
MLP top-5	1.000	1.000	1.000	0.999	0.996	0.992	0.996	0.994	0.989

Routing ids alone, over all 24 layers, give top-1 0.54 with an MLP; the pair of experts chosen at a single layer gives 0.007 to 0.017. Coverage depends on placement: a worker that holds one expert index in all layers sees 99.7% to 99.95% of positions, a worker that holds one layer's expert sees 24% to 27%, and eight random (layer, expert) pairs see 91%. Even a worker holding one (layer, expert) pair attributes the source genome with accuracy 0.68 to 0.73 from 30 tokens. Table 9 shows why conventional mitigations do not replace coding.

D.2 Coded experts

With two colluding workers (T=2, N=4), the bound at pad scale 1000 is 3.0 bits per token per forward pass, and at 10^4 it is 0.03 bits.

Browser fidelity of the coded routed path against the plaintext browser path over eight windows: KL 0.0055, dNLL -0.0014, top-1 agreement 0.952, router top-2 agreement 0.985. In the crash test, the WebSocket of the killed worker closed 5 ms after the kill, two pending requests failed and were not needed, and all 16 steps completed.

Table 9: Conventional mitigations. Noise: Gaussian noise of s times the per-dimension standard deviation added to every expert input of all 24 layers; the probe is retrained on noised data. Local layers: experts of the first k layers stay at the coordinator; the entry is the best probe on the remaining layers.

Mitigation	dNLL	Probe top-1, layer 0	Probe top-1, layer 12
Noise s=0.03	+0.0010	1.000	0.934
Noise s=0.1	+0.0184	1.000	0.932
Noise s=0.3	+0.1834	1.000	0.908
		Best remaining layer	
First 2 layers local	0	0.999	
First 8 layers local	0	0.957	

Table 10: Utility of coded experts over the 48 evaluation windows (47,487 tokens), against the same forward pass without coding (mean NLL 4.3333). Every expert of every layer is coded in both rounds.

K	T	N	Pad scale	Decode subset	dNLL	KL	Top-1
1	1	3	0	$\{0, 1\}$	+0.0004	0.0028	0.961
1	1	3	1	$\{0, 1\}$	+0.0003	0.0029	0.961
1	1	3	10	$\{0, 1\}$	+0.0000	0.0028	0.961
1	1	3	100	$\{0, 1\}$	+0.0006	0.0029	0.961
1	1	3	1000	$\{0, 1\}$	+0.0004	0.0030	0.960
1	1	3	1000	$\{1, 2\}$	+0.0006	0.0032	0.960
1	1	3	10^4	$\{0, 1\}$	+0.0010	0.0043	0.950
2	1	3	1000	$\{0, 1, 2\}$	−0.0002	0.0032	0.959
1	2	4	1000	$\{0, 1, 2\}$	+0.0005	0.0033	0.959

Table 11: Adaptive attacker on shares (K=1, T=1, N=3): token top-1 on held-out organisms after retraining on shares with fresh pads, and the information bound of one worker (bits per token per forward pass).

Pad scale	L0 linear	L0 MLP	L12 linear	L12 MLP	MI bound
0	1.000	1.000	0.575	0.938	
1	1.000	1.000	0.486	0.801	434,570
10	0.957	0.924	0.011	0.016	9,066
100	0.007	0.008	0.003	0.005	92.4
1000					0.92
10^4					0.009

Table 12: Coded experts in Chrome: three workers, each hosting all 192 Q4_0 experts, on one H100, pad scale 1000. Prefill of 128 tokens (medians of 4 repetitions; 2 for the full mode) and decode (medians of 15 steps; 7 for the full mode). Plaintext was measured interleaved with each coded mode in the same session. Traffic is measured at the coordinator for coded modes and computed from row sizes for plaintext.

Mode	Visible to a worker	Prefill (s)	Decode (ms)	MB per decode token
Plaintext	plaintext rows	0.41	92.8	0.6
Coded, routed	shares, routing	6.34	220.4	17.7
Coded, dense	shares, round-two routing	15.92	300.1	45.1
Coded, full	shares only	22.41	375.6	64.6