A Typed Tensor Language for Shared-State Federated Computation

Theofilos Mailis[1,2*] **Theodore Papamarkou**[1,3*] **Andreas Ktenidis**[1,4]
Kalliopi-Christina Despotidou[1,2] **Konstantinos Filippopolitis**[1] **Yannis Foufoulas**[1]
Thanasis-Michail Karampatsis[1] **Evdokia Mailli**[1] **Yannis Ioannidis**[1,2]

[1]Athena Research Center [2]National and Kapodistrian University of Athens
[3]National Technical University of Athens [4]Athens University of Economics and Business

Abstract

Shared-state federated computations combine client-local tensor computation, mergeable aggregation into shared state, and shared-only post-processing. We introduce a typed tensor language for this class of computations. Its two tensor sorts separate client-partitioned data from globally available values, and typing tracks the partitioned axis. A virtual global tensor serves as a semantic reference for centralized evaluation. We show that typed one-round programs factor through shared tensors whose shapes depend on the program but are independent of client and sample counts. The converse applies to typed-realizable factorizations: each encoder component is represented by an allowed aggregation or contraction with its valid merge, and the decoder is shared-only. The construction extends round by round to programs whose persistent state is shared. For a loss supplied with a client-local per-sample gradient expression, summation represents the empirical gradient. This gives typed programs for server-side first-order updates and, with shared linear algebra, curvature-block updates. The language covers federated analytics and FedSGD. General multi-local-step FedAvg and persistent private client state are outside its scope.

1 Introduction

Federated learning is usually introduced through a simple constraint: data are stored across clients, and raw records should not be centralized. The standard response is to let clients compute local quantities and let a server aggregate them into shared state. This pattern appears in gradient-based learning, second-order optimization, federated analytics, and classical statistical summaries such as means, covariances, normal-equation blocks, and curvature matrices.

Despite this common structure, federated computations are often specified algorithm by algorithm. A federated mean, gradient, covariance, optimizer update, or Newton block may be written as a separate protocol, even when they share the same mathematical form. This obscures a basic question: which tensor computations over distributed records can be expressed as client-local computation, followed by fixed-dimensional shared state, followed by shared-only post-processing?

This paper studies shared-state federated computation. Inter-client dependence passes through shared tensors whose shapes are independent of client and sample counts. The language agrees with evaluation on a virtual global tensor, used as a semantic reference. It covers means and covariances, FedSGD [McMahan et al., 2017], and server-side optimization. General multi-local-step FedAvg [McMahan et al., 2017] and persistent private client state are outside the language's scope.

*Equal contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

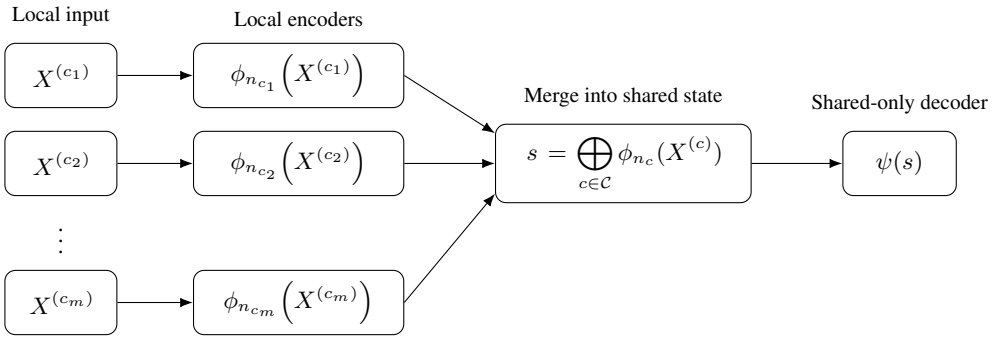

Figure 1: Shared-state factorization of a federated computation. Each client applies a local encoder to its local tensor, the encoded summaries are merged into a finite-dimensional shared state, and a shared-only decoder produces the final output.

Its two tensor sorts separate client-partitioned data from shared values, and typing tracks the axis along which the data are partitioned.

Why record-axis typing matters. Existing systems provide placement-aware programming interfaces for federated computation [Beutel et al., 2020, TensorFlow, 2021]. Our focus is complementary; *we track the tensor-level record axis and make clear when a computation remains client-local and when it produces shared state. This rules out expressions that treat a federated record axis as if it were already globally available, unless the computation passes through a designated record-axis elimination step.* For example, a normal-equation block such as $X^\top X$ is represented as $\sum_{c\in\mathcal{C}} {X^{(c)}}^\top X^{(c)}$, not as an ordinary shared matrix product on a global tensor. *The point is not to identify bugs in a particular framework, but to give formal tensor semantics in which locality, exposure, and centralized equivalence are visible in the type structure.*

Shared-state factorization. The central semantic object is a shared-state factorization. Let $\mathcal{C}$ denote the set of clients, let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ be a federated input, and let n_c be the local number of records at client c. A federated computation admits a shared-state factorization if it can be written as $\sigma_{\mathcal{C}}(X) = \psi\left(\bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})\right)$, where each encoder ϕ_{n_c} is evaluated locally on client c, $\oplus$ is a merge operation on fixed-dimensional client summaries, and ψ is a shared-only decoder. Figure 1 visualizes this three-stage structure of local encoding, aggregation (merging) into shared state, and shared-only decoding. The shared state is finite-dimensional and fixed in advance; it may depend on the computation, but not on the number of clients or on their local record counts.

Contributions. The three-step pattern of computing local summaries, aggregating them, and decoding the result has appeared in federated learning, database aggregation, and mergeable summaries [Gray et al., 1997, Agarwal et al., 2013, McMahan et al., 2017]. Our contribution is to give this pattern a typed federated tensor-language formulation, connect it to virtual-global semantics, and prove expressibility and factorization theorems for the shared-state program classes defined in Section 3. More specifically, our contributions are as follows.

1. *Language definition and semantics*. We define a typed federated tensor language with federated and shared tensors, record-axis-aware typing, and virtual-global semantics. The language ensures client-locality, restricts how shared outputs may arise from federated inputs, and agrees with centralized evaluation on the virtual global tensor.
2. *Factorization theorems and iteration*. We prove structural expressibility theorems for one-round and iterative programs. Each one-round program induces a shared-state factorization, and each typed-realizable factorization is represented by such a program. Iterative programs retain shared state between rounds.
3. *Differentiable learning and optimization*. We specialize the framework to differentiable learning. Representably differentiable losses induce federated gradient tensors whose record-axis sums yield global gradients. This gives typed programs for server-side gradient descent, momentum, and Adam, and, after adding shared-only linear algebra primitives, curvature-block methods such as damped Newton, Gauss-Newton, and iteratively reweighted least squares (IRLS) whenever their updates are shared-only functions of aggregated gradients and curvature blocks.

Scope. Uniform client encoders and commutative merging exclude outputs that depend on client order. Representing median and quantile operators in our language requires a typed mergeable state of bounded dimension. Kernel methods retaining state whose dimension grows with sample counts fall outside our language's scope. A transient local-encoder interface expresses multi-step FedAvg/FedProx with round-wise factorization without virtual-global consistency. Persistent private state requires further semantics. Exposure discipline restricts shared-state formation and gives no privacy guarantee. Appendix L presents Federated Tensor Algebra [FeTA; Foufoulas et al., 2026], which is our Python package implementation, and associated reported code-complexity results.

Paper structure. The rest of the paper is organized as follows. Section 2 introduces the typed federated tensor language and its semantic guarantees. Section 3 proves the shared-state factorization results. Section 4 specializes the theory to differentiable losses and learning updates. Section 5 discusses limitations and extensions. Related work is discussed in Appendix A.

2 Typed federated tensor language

This section defines the base typed tensor language. The language separates federated tensors from shared tensors, tracks the record axis of federated data, and gives virtual-global semantics for comparing distributed execution with centralized tensor evaluation. We then state the three semantic guarantees used throughout the paper, namely client-locality, exposure discipline, and virtual-global consistency.

2.1 Semantic objects

We work in a federated setting. Each client stores data that are partitioned along one tensor axis, called the *record axis*. Shared tensors are available at every client. The *virtual global tensor* is a centralized reference object obtained by concatenating client-local tensors along the record axis in a fixed client order. It is used only for semantics.

Definition 1 (Federation)**.** *A federation is a finite nonempty set $\mathcal{C}$ equipped with a fixed total order. We write $\mathcal{C} = \{c_1, \dots, c_m\}$, $c_1 < \dots < c_m$.*

For each integer $k \geq 1$, we write $[k] := \{1, \dots, k\}$ and $\mathbb{N}_0 := \{0, 1, 2, \dots\}$. We also set $[0] := \varnothing$. Scalars are identified with 0-way tensors.

Definition 2 (Federated tensor)**.** *Let $k \geq 1$, let $r \in [k]$, and let $\mathbf{d} = (d_1, \dots, d_{r-1}, d_{r+1}, \dots, d_k)$ be a tuple of positive integers. When $k = 1$, the tuple $\mathbf{d}$ is empty. A k-way federated tensor with record axis r and common non-record shape $\mathbf{d}$ is a family $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ for which there exists a family of local record counts $\{n_c\}_{c \in \mathcal{C}} \subseteq \mathbb{N}_0$ such that $X^{(c)} \in \mathbb{R}^{d_1 \times \dots \times d_{r-1} \times n_c \times d_{r+1} \times \dots \times d_k}$ for every $c \in \mathcal{C}$.*

Definition 3 (Shared tensor)**.** *A shared tensor is either a scalar $S \in \mathbb{R}$, or a tensor $S \in \mathbb{R}^{s_1 \times \dots \times s_k}$ for some integer $k \geq 1$ and some positive integers $s_1, \dots, s_k$, with the property that the same value of S is available at every client.*

Global availability refers to the shared sort. A federated value retains its client-indexed sort even when there is one client.

Definition 4 (Virtual global tensor)**.** *Let $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ be a federated tensor with record axis r, common non-record shape $\mathbf{d} = (d_1, \dots, d_{r-1}, d_{r+1}, \dots, d_k)$, and local record counts $\{n_c\}_{c \in \mathcal{C}}$. The virtual global tensor of X is $\overline{X} := \mathrm{Concat}_r(X^{(c_1)}, \dots, X^{(c_m)})$, where $\mathcal{C} = \{c_1, \dots, c_m\}$ is listed in increasing order and Concat_r denotes concatenation along axis r. Moreover,*

$$\overline{X} \in \mathbb{R}^{d_1 \times \dots \times d_{r-1} \times \left(\sum_{j=1}^{m} n_{c_j}\right) \times d_{r+1} \times \dots \times d_k}.$$

The fixed order on $\mathcal{C}$ is used to define $\overline{X}$. The tensor $\overline{X}$ is a semantic reference object that is used to state the correctness of typed federated computations. The local record counts $\{n_c\}_{c \in \mathcal{C}}$ are part of the value of a federated tensor.

2.2 Typed expressions and primitive operations

The language has two tensor sorts. A shared type is written $\mathsf{Sh}(\mathbf{s})$, where $\mathbf{s}$ is an ordinary shape tuple. A federated type is written $\mathsf{Fed}_r(\mathbf{d})$, where r is the record axis and $\mathbf{d}$ is the common non-record shape. This is the type-level counterpart of Definition 2.

$$\frac{\Gamma \vdash e : \mathsf{Fed}_r(\mathbf{d})}{\Gamma \vdash \alpha_r(e) : \mathsf{Sh}(\mathbf{d})} \qquad \frac{\Gamma \vdash e : T \quad j \neq r}{\Gamma \vdash \alpha_j(e) : \mathrm{del}_j(T)}$$

$$\frac{\Gamma \vdash a : \mathsf{Fed}_2((u)) \quad \Gamma \vdash b : \mathsf{Fed}_1((v))}{\Gamma \vdash \mathsf{MatMul}_{\mathsf{FedFed}}(a, b) : \mathsf{Sh}((u, v))}$$

Figure 2: Aggregation with $T = \mathsf{Fed}_r(\mathbf{d})$, $\alpha \in \mathcal{A}$, and $j \in [|\mathbf{d}| + 1]$. The first rule removes axis r, the second removes another axis and retains federated output. The contraction requires matching client-local lengths along the contracted axes. The factorization program class also requires the merge structure of Section 3.2.

Expressions are built from variables, corresponding to federated or shared tensors, by applying primitive tensor operations. A typing judgment has the form $\Gamma \vdash e : T$, where Γ assigns tensor types to variables, e is an expression, and T is either shared or federated. The base primitive signature is denoted by Σ_0. It contains unary element-wise maps, binary element-wise maps, binary comparisons, axis-wise aggregations, axis permutations, and three matrix-product primitives. The full syntax, shape rules, broadcast rules, semantic clauses, and typing rules are given in Appendix B.

The base signature uses scalar unary maps $\mathcal{U}$, binary maps $\mathcal{B}$, comparisons $\mathcal{G}$, aggregation families $\mathcal{A}$, and permutations $\mathcal{P}$. Figure 2 gives the aggregation rules and the federated-federated matrix contraction. The other matrix primitives, $\mathsf{MatMul}_{\mathsf{FedSh}}$ and $\mathsf{MatMul}_{\mathsf{ShFed}}$, retain federated output. Full shape and compatibility conditions appear in Appendix B.

Example 1 (A federated mean)**.** *For scalar samples, choose* $\mathrm{Sum} \in \mathcal{A}$*, the map* $u_1(z) = 1$ *in* $\mathcal{U}$*, and the total division map below in* $\mathcal{B}$*:*

$$\phi_n(z) = \left(\sum_{a=1}^{n} z_a, n \right), \ \oplus = +, \ \psi(A, N) = \begin{cases} A/N, & N > 0, \\ 0, & N \leq 0. \end{cases}$$

Both components are typed sums, the second summing $u_1(z)$*. An empty client sends* $(0, 0)$*. For a positive total count the decoder returns the global mean; its other branch makes it total on* $\mathbb{R}^2$*.*

We use the following terminology throughout the paper. A typed primitive application *is an application of a primitive symbol in* Σ_0 *that satisfies the corresponding typing, shape, broadcast, and record-axis conditions of Appendix B. A typed expression* $\Gamma \vdash e : T$ *is a* shared expression *if* T *is a shared type, and it is a* federated expression *if* T *is a federated type.*

A federated expression is called *client-local* if every primitive subexpression that receives at least one federated-typed argument has federated type. Thus, client-local expressions may use shared inputs, but they do not perform an intermediate operation that converts federated data into shared state. Record-axis aggregations and the federated-federated matrix product are therefore excluded from client-local expressions, except when they are the shared-output step being studied. A shared expression is called *shared-only* if all free variables in its typing context have shared type. Shared-only expressions model post-processing performed after the shared state has been formed.

The main role of the type system is to separate local computation from shared-state formation. Client-local federated expressions preserve the record axis and can be evaluated independently on each client. Shared state is formed only when the record axis is eliminated by a designated shared-output primitive, such as record-axis aggregation or the federated-federated matrix product. Exposure discipline controls which operations form shared values, without providing differential privacy, cryptographic confidentiality, or protection against inference from those values. The next subsection records the three semantic guarantees that make this separation precise.

2.3 Foundational semantic propositions

The full formal specification of the base typed language is given in Appendix B. In this subsection, a typed primitive application means an application of a primitive symbol of Σ_0 satisfying the typing, shape, broadcast, and axis conditions of that specification.

For each primitive symbol F of Σ_0, we write F^{ord} for its ordinary centralized interpretation. If F is one of the symbols in $\mathcal{U} \cup \mathcal{B} \cup \mathcal{G} \cup \{\alpha_j : \alpha \in \mathcal{A}, \ j \geq 1\} \cup \mathcal{P}$, then F^{ord} is the corresponding

ordinary tensor operation. If F is one of the three matrix-product symbols, then F^{ord} is ordinary matrix multiplication in the displayed operand order.

Proposition 1 states client-locality of every federated-output primitive application. Proposition 2 identifies the primitive applications that can produce shared state from federated inputs. Proposition 3 states agreement between the distributed primitive semantics and centralized evaluation on the virtual global tensor.

Proposition 1 (Client-locality)**.** *Let F be a primitive symbol of Σ_0. Consider a typed primitive application of F whose federated inputs are $X_1, \ldots, X_p$ and whose shared inputs are $S_1, \ldots, S_q$, and let Y denote its output. Assume that Y is a federated tensor. Then, for every client $c \in \mathcal{C}$, there exists a deterministic map $\Phi_{F,c}$ such that $Y^{(c)} = \Phi_{F,c}(X_1^{(c)}, \ldots, X_p^{(c)}, S_1, \ldots, S_q)$. The maps $\Phi_{F,c}$ can be chosen so that they depend on the primitive symbol, the operand positions, and the local input shapes at client c, but not on the identity of c or on any tensor stored at another client.*

Proposition 2 (Exposure discipline)**.** *Let F be a primitive symbol of Σ_0. Consider a typed primitive application of F whose federated inputs are $X_1, \ldots, X_p$ and whose shared inputs are $S_1, \ldots, S_q$, and let Y denote its output. The output object is globally available if and only if Y is a shared tensor. Assume that $p \geq 1$ and that Y is shared. Then exactly one of the following two cases holds. There exist $\alpha \in \mathcal{A}$, an index $i \in [p]$, and the record axis r of X_i such that $Y = \alpha_r(X_i)$. There exist indices $i, j \in [p]$ with $i \neq j$ such that $Y = \mathsf{MatMul}_{\mathsf{FedFed}}(X_i, X_j)$, where $\mathsf{MatMul}_{\mathsf{FedFed}}$ is the federated-federated matrix-product primitive that contracts the record axes of two federated matrix inputs and returns a shared matrix (see Appendix B).*

Proposition 3 (Virtual-global consistency)**.** *Let F be a primitive symbol of Σ_0 with arity m. Consider a typed primitive application of F to inputs $A_1, \ldots, A_m$, where each A_t is either a federated tensor or a shared tensor, and let Y denote its output. For each $t \in [m]$, define*

$$\widetilde{A}_t = \begin{cases} \overline{A_t}, & \text{if } A_t \text{ is a federated tensor}, \\ A_t, & \text{if } A_t \text{ is a shared tensor}. \end{cases}$$

If Y is a federated tensor, then $\overline{Y} = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m)$. If Y is a shared tensor, then $Y = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m)$.

Propositions 1, 2, and 3 complete the setup of the base typed language, and form the semantic input to the shared-state factorization theorems developed in Section 3. Their proofs are given in Appendix C.

3 Shared-state factorizations

The semantic propositions of Section 2 show that the base language separates client-local computation from shared-state formation. We now use this separation to describe the class of shared-output computations represented by the language. The central object is a factorization through a fixed-dimensional shared state. The state is computed locally, merged across clients, and decoded into the final shared output. The proofs of the theoretical statements of this section are given in Appendix D.

3.1 Semantic shared-state factorizations

Consider an input type $\mathsf{Fed}_r(\mathbf{d})$, where $\mathbf{d} = (d_1, \ldots, d_{r-1}, d_{r+1}, \ldots, d_k)$, and an output shape $\mathbf{t}$. For each $n \in \mathbb{N}_0$, write $\mathcal{X}_n(r, \mathbf{d}) := \mathbb{R}^{d_1 \times \ldots \times d_{r-1} \times n \times d_{r+1} \times \ldots \times d_k}$. A shared-output computation of input type $\mathsf{Fed}_r(\mathbf{d})$ and output shape $\mathbf{t}$ is a rule σ that assigns to every finite ordered federation $\mathcal{C}$ and every federated tensor $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$ an output $\sigma_{\mathcal{C}}(X) \in \mathbb{R}^{\mathbf{t}}$.

Definition 5 (Shared-state factorization)**.** *Let σ be a shared-output computation of input type $\mathsf{Fed}_r(\mathbf{d})$ and output shape $\mathbf{t}$. A shared-state factorization of σ consists of a finite product of tensor spaces $M = \mathbb{R}^{\mathbf{s}_1} \times \ldots \times \mathbb{R}^{\mathbf{s}_q}$, where $q \geq 1$, a binary operation $\oplus_M : M \times M \to M$, an element $0_M \in M$, local encoder maps $\phi_n : \mathcal{X}_n(r, \mathbf{d}) \to M$ for every $n \in \mathbb{N}_0$, and a decoder map $\psi : M \to \mathbb{R}^{\mathbf{t}}$, such that the following two conditions hold. First, $(M, \oplus_M, 0_M)$ is a commutative monoid. Thus, for all $a, b, c \in M$, we have $(a \oplus_M b) \oplus_M c = a \oplus_M (b \oplus_M c)$, $a \oplus_M b = b \oplus_M a$, and $a \oplus_M 0_M = a$. Second, for every finite ordered federation $\mathcal{C}$ and every federated tensor $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$ with local record counts $\{n_c\}_{c \in \mathcal{C}}$, we have $\sigma_{\mathcal{C}}(X) = \psi\left(\bigoplus_{c \in \mathcal{C}} \phi_{n_c}(X^{(c)})\right)$.*

The finite product M is independent of the federation and of the local record counts. This is the fixed-dimensional shared state of the computation. Definition 5 is semantic and does not require the encoders or decoder to be expressed in the typed language. The next subsection connects this semantic factorization with typed one-round programs.

The same factorization also identifies the tensor-level interface for privacy mechanisms. Local differential privacy may be applied to the encoded client messages, secure multi-party computation may realize the merge, and central differential privacy may be applied to the merged state or decoded output. Appendix E formalizes this observation as a privacy-lifting result for shared-state factorizations.

3.2 One-round typed programs

We restrict the semantic factorization of Definition 5 to factorizations generated by typed expressions. A one-round program first evaluates finitely many client-local federated expressions. It then converts each of them into shared state by an allowed shared-output primitive. Finally, it applies a shared-only decoder. Throughout this subsection, expressions are over the base primitive signature Σ_0.

For an aggregation α_r with output shape $\mathbf{s}$, a *merge structure* is a commutative monoid $(\mathbb{R}^{\mathbf{s}}, \oplus, 0_{\alpha,\mathbf{s}})$ satisfying

$$\alpha_r(U \parallel_r V) = \alpha_r(U) \oplus \alpha_r(V),\ \alpha_r(E_{r,\mathbf{s}}) = 0_{\alpha,\mathbf{s}},$$

for all tensors U, V with non-r shape $\mathbf{s}$, including empty blocks; $\|_r$ denotes concatenation on axis r, and $E_{r,\mathbf{s}}$ has this non-r shape and extent zero on axis r. The identity can differ from the numerical zero tensor. Sum uses addition and zero. Product uses elementwise multiplication and the all-ones tensor, when included in the signature. Merge structures are required for aggregation boundaries in the one-round and iterative typed program classes (Definitions 6 and 7).

A shared-state factorization is *typed-realizable* when its component encoders have the valid aggregation or aligned contraction forms used below, and its decoder has a shared-only realization on the whole state space. Definition 41 gives the realization witnesses.

Definition 6 (One-round typed program)**.** *Consider an input type $\mathsf{Fed}_r(\mathbf{d})$ and an output shape $\mathbf{t}$. A one-round typed program consists of shared-state expressions $g_1(x), \dots, g_q(x)$ and a shared-only decoder expression h, where $q \geq 1$, with the following properties.*

For each $i \in [q]$, the expression $g_i(x)$ has one of the following two forms. The first form is record-axis aggregation. There are a client-local federated expression $e_i(x)$, a type $\mathsf{Fed}_{r_i}(\mathbf{u}_i)$, and an aggregation schema $\alpha^{(i)} \in \mathcal{A}$ such that $x : \mathsf{Fed}_r(\mathbf{d}) \vdash e_i(x) : \mathsf{Fed}_{r_i}(\mathbf{u}_i)$ and $g_i(x) = \alpha^{(i)}_{r_i}(e_i(x))$. The output shape of $g_i(x)$ is denoted by $\mathbf{s}_i$. The aggregation has a merge structure $(\mathbb{R}^{\mathbf{s}_i}, \oplus_i, 0_i)$. Its concatenation law gives $\alpha^{(i)}_{r_i}(Z) = \bigoplus_{c \in \mathcal{C}} \alpha^{(i)}_{r_i}(Z^{(c)})$ for every federated tensor Z of the indicated type. The second form is a federated-federated matrix product. There are client-local federated expressions $a_i(x)$ and $b_i(x)$ and positive integers m_i, n_i such that $x : \mathsf{Fed}_r(\mathbf{d}) \vdash a_i(x) : \mathsf{Fed}_2((m_i))$ and $x : \mathsf{Fed}_r(\mathbf{d}) \vdash b_i(x) : \mathsf{Fed}_1((n_i))$, and $g_i(x) = \mathsf{MatMul}_{\mathsf{FedFed}}(a_i(x), b_i(x))$. The two operands must satisfy the client-wise count compatibility condition of Definition 28. In this case, $\mathbf{s}_i = (m_i, n_i)$ and the associated merge operation on $\mathbb{R}^{\mathbf{s}_i}$ is ordinary addition.

The decoder is a shared-only expression $y_1 : \mathsf{Sh}(\mathbf{s}_1), \dots, y_q : \mathsf{Sh}(\mathbf{s}_q) \vdash h(y_1, \dots, y_q) : \mathsf{Sh}(\mathbf{t})$. The shared-output computation induced by the program is $\sigma_{\mathcal{C}}(X) = h(g_1(X), \dots, g_q(X))$ for every finite ordered federation $\mathcal{C}$ and every federated input X of type $\mathsf{Fed}_r(\mathbf{d})$.

The two forms of g_i are the shared-output mechanisms from federated inputs in Proposition 2. The merge structure restricts aggregation boundaries to client-wise merges whose output shape is independent of client and sample counts. The next result gives a semantic shared-state factorization for each one-round typed program.

Theorem 1 (One-round shared-state factorization)**.** *Let σ be the shared-output computation induced by a one-round typed program of input type $\mathsf{Fed}_r(\mathbf{d})$ and output shape $\mathbf{t}$. Then σ admits a shared-state factorization (see Definition 5).*

The construction uses structural induction for uniform local realizations (Proposition 1), the permitted shared-output boundaries (Proposition 2), and their product merge. Shared-only decoding completes the factorization; Proposition 3 relates base execution to ordinary tensor evaluation. Iteration repeats the construction with the current shared parameters.

Proposition 4 (Effective factorization extraction)**.** *From a valid merge-annotated derivation $\mathcal{D}$ of a one-round typed program, the factorization of Theorem 1 can be extracted in $O(|\mathcal{D}|)$ time and represented in $O(|\mathcal{D}|)$ space, with shared-subexpression DAG output and constant-time primitive metadata lookup.*

Corollary 1 (Extracted state dimension)**.** *For $\mathbf{s} = (s_1, \ldots, s_k)$, let $\nu(\mathbf{s}) = \prod_{j=1}^{k} s_j$, with $\nu(()) = 1$. The product state extracted from a one-round program P has dimension*

$$D(P) := \dim M = \sum_{i=1}^{q} \nu(\mathbf{s}_i).$$

Each declared shared-state component contributes to the sum appearing in $D(P)$, including scalars and repeated components. The quantity $D(P)$ is a constructive upper bound for representing the computation in real coordinates, without a claim of minimum dimension or communication optimality. Bounds on communication in bits and statistical accuracy, such as those of Braverman et al. [2016], Zhang et al. [2013], use task and protocol assumptions. Comparison requires an accuracy criterion and a finite-precision protocol model, which we leave for future work (Appendix M).

The converse concerns the typed-realizable class of Definition 41.

Theorem 2 (One-round realization of shared-state factorizations)**.** *Every typed-realizable shared-state factorization is induced by a one-round typed program.*

The two theorems characterize the one-round class with the specified encoder, merge and decoder forms. The class is closed under shared-only post-processing, as used for means and variances (Appendix F). It covers typed realizations of the distributive and algebraic aggregates of Gray et al. [1997]. Representing median or quantile operators in our typed tensor language requires a typed mergeable sufficient state with shape independent of client and sample counts. Finitely many real coordinates, without restrictions on their encoding, do not establish an impossibility result for such operators.

3.3 Iterative typed programs

Learning algorithms usually produce a sequence of shared states rather than a single summary. The shared state may be a parameter vector, a collection of optimizer moments, a curvature approximation, or a nonempty finite tuple of shared tensors. The same factorization principle applies round by round. Each round uses the current shared state inside client-local expressions, forms a finite shared state by record-axis elimination, and then applies a shared-only update map.

A tuple with component shapes $\mathbf{s}_1, \ldots, \mathbf{s}_q$ can be stored as a shared vector of length $\sum_{j=1}^{q} \nu(\mathbf{s}_j)$ by concatenating flattened components. Packing and component extraction are total shared-only primitives of a conservative extension (Appendix K). Extraction selects the component's block and restores its shape.

Definition 7 (Iterative typed program)**.** *Fix an input type $\mathsf{Fed}_r(\mathbf{d})$, an integer $T \geq 1$, and shared-state shapes $\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_T$. An iterative typed program consists of an initial shared state $\theta_0 \in \mathbb{R}^{\boldsymbol{\tau}_0}$ and, for each round $t \in \{0, \ldots, T-1\}$, shared-state expressions $g_{t,1}(x, \theta), \ldots, g_{t,q_t}(x, \theta)$ and a shared-only decoder expression h_t, where $q_t \geq 1$, with the following properties.*

For each $i \in [q_t]$, the expression $g_{t,i}(x, \theta)$ has one of the two forms allowed in Definition 6, with the typing context enlarged by the current shared state $\theta : \mathsf{Sh}(\boldsymbol{\tau}_t)$. In the aggregation form, there are a client-local federated expression $e_{t,i}(x, \theta)$, a type $\mathsf{Fed}_{r_{t,i}}(\mathbf{u}_{t,i})$, and an aggregation schema $\alpha^{(t,i)} \in \mathcal{A}$ such that $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash e_{t,i}(x, \theta) : \mathsf{Fed}_{r_{t,i}}(\mathbf{u}_{t,i})$ and $g_{t,i}(x, \theta) = \alpha^{(t,i)}_{r_{t,i}}(e_{t,i}(x, \theta))$. The output shape of $g_{t,i}$ is denoted by $\mathbf{s}_{t,i}$. The aggregation has a merge structure $(\mathbb{R}^{\mathbf{s}_{t,i}}, \oplus_{t,i}, 0_{t,i})$, so that $\alpha^{(t,i)}_{r_{t,i}}(Z) = \bigoplus_{c \in \mathcal{C}} \alpha^{(t,i)}_{r_{t,i}}(Z^{(c)})$ for every federated tensor Z of the indicated type. In the matrix-product form, there are client-local federated expressions $a_{t,i}(x, \theta)$ and $b_{t,i}(x, \theta)$ and positive integers $m_{t,i}, n_{t,i}$ such that $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash a_{t,i}(x, \theta) : \mathsf{Fed}_2((m_{t,i}))$ and $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash b_{t,i}(x, \theta) : \mathsf{Fed}_1((n_{t,i}))$. In this case, $g_{t,i}(x, \theta) = \mathsf{MatMul}_{\mathsf{FedFed}}(a_{t,i}(x, \theta), b_{t,i}(x, \theta))$, $\mathbf{s}_{t,i} = (m_{t,i}, n_{t,i})$, and the associated merge operation on $\mathbb{R}^{\mathbf{s}_{t,i}}$ is ordinary addition. The contraction satisfies the same client-wise count compatibility condition as in Definition 6.

The round-t decoder is a shared-only expression $y_1 : \mathsf{Sh}(\mathbf{s}_{t,1}), \ldots, y_{q_t} : \mathsf{Sh}(\mathbf{s}_{t,q_t}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash h_t(y_1, \ldots, y_{q_t}, \theta) : \mathsf{Sh}(\boldsymbol{\tau}_{t+1})$.

For a finite ordered federation $\mathcal{C}$ and a federated input X of type $\mathsf{Fed}_r(\mathbf{d})$, the program induces shared iterates by $\theta_{t+1} = h_t(g_{t,1}(X,\theta_t),\dots,g_{t,q_t}(X,\theta_t),\theta_t)$ for $t = 0,\dots,T-1$. The induced shared-output computation is $\sigma_{\mathcal{C}}(X) := \theta_T$.

Definition 7 does not introduce private state that persists across rounds. The only state carried from one round to the next is the shared state θ_t. This matches algorithms whose communication pattern is server-side state updated from client-local tensor computations and shared aggregation.

Theorem 3 (Iterative shared-state factorization)**.** *Let σ be induced by an iterative typed program of input type $\mathsf{Fed}_r(\mathbf{d})$ and shared-state shapes $\boldsymbol{\tau}_0,\dots,\boldsymbol{\tau}_T$. For each round $t \in \{0,\dots,T-1\}$, define the round update $U_{t,\mathcal{C}}(X,\theta) := h_t(g_{t,1}(X,\theta),\dots,g_{t,q_t}(X,\theta),\theta)$. Then, for each $t \in \{0,\dots,T-1\}$, there exist a finite product of tensor spaces M_t, a commutative monoid operation $\oplus_t$ on M_t, an identity element $0_t \in M_t$, local encoder maps $\phi_{t,n}(\cdot;\theta) : \mathcal{X}_n(r,\mathbf{d}) \to M_t$ for every $n \in \mathbb{N}_0$ and every $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$, and a decoder map $\psi_t : M_t \times \mathbb{R}^{\boldsymbol{\tau}_t} \to \mathbb{R}^{\boldsymbol{\tau}_{t+1}}$ such that $U_{t,\mathcal{C}}(X,\theta) = \psi_t\big(\bigoplus_{c\in\mathcal{C}} \phi_{t,n_c}(X^{(c)};\theta),\theta\big)$ for every finite ordered federation $\mathcal{C}$, every federated tensor $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$ with local record counts $\{n_c\}_{c\in\mathcal{C}}$, and every shared state $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$.*

Theorem 3 is the round-wise analogue of Theorem 1. It shows that iterative typed programs do not introduce a new communication principle. Each round is still an encode, merge, and decode computation through fixed-dimensional shared state.

For the one-round and iterative program classes, the language and its conservative extensions retain uniform locality, the designated shared-output boundaries, factorization and virtual-global consistency, with only shared state across rounds (Appendix G). A separate transient-state interface allows local optimizer steps and messages of count-independent shape, retaining factorization and exposure through the designated merge but producing partition-dependent updates; this covers multi-step FedAvg and FedProx (Appendix H). Persistent private client variables, as in SCAFFOLD [Karimireddy et al., 2020] or FedDyn [Acar et al., 2021], require a further state sort and cross-round semantics beyond this paper.

4 Differentiable fragment and learning updates

The factorization results of Section 3 apply to the one-round and iterative typed program classes. We now specialize them to learning updates. The basic case is an empirical objective whose per-record loss is computed locally and whose gradient is obtained by summing per-record gradients across clients. This gives a direct route from the typed language to first-order federated optimization. The proofs of the theoretical statements of this section are given in Appendix I.

Throughout this section, assume that $\mathcal{A}$ contains ordinary summation, denoted by Sum. If $\boldsymbol{\tau}$ is a shape tuple, we identify $\mathbb{R}^{\boldsymbol{\tau}}$ with a Euclidean space using the standard coordinate inner product. For a differentiable map $f : \mathbb{R}^{\boldsymbol{\tau}} \to \mathbb{R}$, we write $\nabla_\theta f(\theta) \in \mathbb{R}^{\boldsymbol{\tau}}$ for its gradient.

4.1 Representably differentiable losses and gradient expressibility

Let $\mathsf{Fed}_r(\mathbf{d})$ and $\mathsf{Sh}(\boldsymbol{\tau})$ be an input type and a shared parameter type, respectively. A per-record scalar loss is represented in the language by a client-local federated expression $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash \ell(x,\theta) : \mathsf{Fed}_1(\varnothing)$. Thus, for a client with n records, the local value of ℓ is a vector in $\mathbb{R}^n$. A per-record parameter-gradient expression has type $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash \gamma(x,\theta) : \mathsf{Fed}_1(\boldsymbol{\tau})$. For a client with n records, the local value of γ is a tensor in $\mathbb{R}^{n\times\boldsymbol{\tau}}$, with one gradient tensor of shape $\boldsymbol{\tau}$ for each record.

Definition 8 (Representably differentiable loss)**.** *A pair of client-local federated expressions (ℓ,γ) with the typings above is called a representably differentiable loss if the following condition holds. For every $n \in \mathbb{N}_0$, every local input $Z \in \mathcal{X}_n(r,\mathbf{d})$, and every $\theta \in \mathbb{R}^{\boldsymbol{\tau}}$, let $\widehat{\ell}_n(Z;\theta) \in \mathbb{R}^n$ and $\widehat{\gamma}_n(Z;\theta) \in \mathbb{R}^{n\times\boldsymbol{\tau}}$ denote the local realizations of ℓ and γ. Then, if $n \geq 1$, for each record index $a \in [n]$, the map $\theta \mapsto \widehat{\ell}_n(Z;\theta)[a]$ is differentiable with $\widehat{\gamma}_n(Z;\theta)[a,\cdot] = \nabla_\theta \widehat{\ell}_n(Z;\theta)[a]$.*

Definition 8 separates two issues. The expression ℓ computes the per-record loss, while γ represents the corresponding per-record gradient as an expression in the typed language. The gradient expression

γ is provided with the loss; the present theory does not construct it by automatic differentiation. Theorem 4 only needs the existence of such an expression.

Let (ℓ, γ) be a representably differentiable loss of input type $\mathsf{Fed}_r(\mathbf{d})$ and parameter type $\mathsf{Sh}(\boldsymbol{\tau})$. For a finite ordered federation $\mathcal{C}$, a federated input $X = \{X^{(c)}\}_{c\in\mathcal{C}}$, and a shared parameter $\theta \in \mathbb{R}^{\boldsymbol{\tau}}$, define the empirical objective $L_{\mathcal{C}}(X,\theta) := \sum_{c\in\mathcal{C}} \sum_{a=1}^{n_c} \widehat{\ell}_{n_c}(X^{(c)};\theta)[a]$ and $G_{\mathcal{C}}(X,\theta) := \sum_{c\in\mathcal{C}} \sum_{a=1}^{n_c} \widehat{\gamma}_{n_c}(X^{(c)};\theta)[a,\cdot] \in \mathbb{R}^{\boldsymbol{\tau}}$, where n_c is the local record count.

Theorem 4 (Gradient expressibility)**.** *Let (ℓ,γ) be a representably differentiable loss. For every finite ordered federation $\mathcal{C}$ and every federated input X of type $\mathsf{Fed}_r(\mathbf{d})$, the map $\theta \mapsto L_{\mathcal{C}}(X,\theta)$ is differentiable and $\nabla_\theta L_{\mathcal{C}}(X,\theta) = G_{\mathcal{C}}(X,\theta)$. Moreover, $G_{\mathcal{C}}(X,\theta)$ is represented by the shared-state expression $g(x,\theta) := \mathrm{Sum}_1(\gamma(x,\theta))$, where the summation is along the record axis of $\gamma(x,\theta)$. Thus, $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash g(x,\theta) : \mathsf{Sh}(\boldsymbol{\tau})$, and the gradient computation is a round-wise shared-state computation with ordinary addition as its merge operation.*

The theorem gives the basic learning interface of the language. Once the per-record gradient is expressible as a client-local federated tensor, the global gradient is obtained by a single record-axis summation. No additional federated primitive is needed.

4.2 First-order server-side updates

The gradient expression of Theorem 4 can be used inside any shared-only server update. This includes gradient descent as the simplest case, and optimizers whose auxiliary variables are shared tensors.

Corollary 2 (Server-side first-order updates)**.** *Let (ℓ,γ) be a representably differentiable loss with parameter type $\mathsf{Sh}(\boldsymbol{\tau})$. Fix $T \geq 1$, shared-state shapes $\boldsymbol{\zeta}_0, \ldots, \boldsymbol{\zeta}_T$, and an initial shared optimizer state $s_0 \in \mathbb{R}^{\boldsymbol{\zeta}_0}$. For each round $t \in \{0,\ldots,T-1\}$, assume that there are shared-only expressions $s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash P_t(s) : \mathsf{Sh}(\boldsymbol{\tau})$ and $g : \mathsf{Sh}(\boldsymbol{\tau}), s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash H_t(g,s) : \mathsf{Sh}(\boldsymbol{\zeta}_{t+1})$. For a finite ordered federation $\mathcal{C}$ and a federated input X of type $\mathsf{Fed}_r(\mathbf{d})$, define the iterates by $s_{t+1} = H_t(G_{\mathcal{C}}(X, P_t(s_t)), s_t)$, $t = 0,\ldots,T-1$. Then the T-step update $X \mapsto s_T$ is induced by an iterative typed program. Hence each round admits a shared-state factorization in the sense of Theorem 3.*

For gradient descent, the optimizer state is the parameter itself, P_t is the identity, and $H_t(g,\theta) = \theta - \eta_t g$. Server-side momentum and Adam are covered whenever the optimizer state is shared and the parameter extraction and update maps are expressible as shared-only tensor expressions. In those cases, the shared optimizer state contains the parameter and the required moment tensors, and the only federated quantity used by the optimizer is the aggregated gradient. For a parameter with p coordinates, this summed-gradient message has $D = p$; momentum and Adam retain $O(p)$ shared server memory. Appendix M separates message size from evaluation cost.

4.3 Shared linear algebra and second-order updates

First-order methods require only an aggregated gradient. Second-order and quasi-second-order methods require additional shared tensors, such as Hessian blocks, Gauss–Newton blocks, or normal-equation matrices. These methods remain within the same framework when the curvature blocks are expressible as record-axis sums of client-local tensor expressions and the linear algebra is performed on shared tensors.

In this subsection, the language may be enlarged by shared-only linear-algebra primitives, such as matrix multiplication, linear-system solve, Cholesky factorization, or log-determinant, on their natural domains. Since these primitives act only on shared tensors, they do not change the client-locality or exposure properties of the federated part of the language. The diagonal shift $C + \lambda I$ denotes a shared-only primitive $\mathrm{AddDiag}_\lambda(C)$, whose ordinary interpretation adds the chosen scalar λ to each diagonal entry. For operations with restricted domains, the factorization uses a total shared extension; the update is recovered on inputs whose required applications remain in the original domain (Appendix K).

Corollary 3 (Curvature-block updates)**.** *Fix a parameter dimension $p \geq 1$. Let (ℓ,γ) be a representably differentiable loss with parameter type $\mathsf{Sh}((p))$. Let $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}((p)) \vdash B(x,\theta) : \mathsf{Fed}_1((p,p))$ be a client-local federated expression. For a finite ordered federation $\mathcal{C}$, a federated input*

X, and a shared parameter $\theta \in \mathbb{R}^p$, define $C_{\mathcal{C}}(X, \theta) := \sum_{c \in \mathcal{C}} \sum_{a=1}^{n_c} \widehat{B}_{n_c}(X^{(c)}; \theta)[a, \cdot, \cdot] \in \mathbb{R}^{p \times p}$, where $\widehat{B}_n$ denotes the local realization of B.

Fix $T \geq 1$, shared-state shapes $\boldsymbol{\zeta}_0, \ldots, \boldsymbol{\zeta}_T$, and an initial shared optimizer state $s_0 \in \mathbb{R}^{\boldsymbol{\zeta}_0}$. For each round $t \in \{0, \ldots, T-1\}$, assume that there are shared-only expressions $s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash P_t(s) : \mathsf{Sh}((p))$ and $g : \mathsf{Sh}((p)), C : \mathsf{Sh}((p,p)), s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash H_t(g, C, s) : \mathsf{Sh}(\boldsymbol{\zeta}_{t+1})$. Define the iterates by $s_{t+1} = H_t(G_{\mathcal{C}}(X, P_t(s_t)), C_{\mathcal{C}}(X, P_t(s_t)), s_t),\ t = 0, \ldots, T-1$. Then the T-step update $X \mapsto s_T$ is induced by an iterative typed program in the shared-linear-algebra extension of the language. Hence each round admits a shared-state factorization in the sense of Theorem 3.

Newton, damped Newton, Gauss–Newton, and IRLS-type updates are instances of Corollary 3 when their curvature or normal-equation blocks are expressible by B. For example, a damped Newton step has shared-only decoder $H_t(g, C, \theta) = \theta - \eta_t \operatorname{Solve}(C + \lambda_t I, g)$ on the domain where $C + \lambda_t I$ is nonsingular. Appendix J gives two learning examples. The dense message (g, C) has $D = p + p^2$. Structured curvature uses smaller messages when represented by suitable mergeable expressions. Otherwise it changes or approximates the update.

The base signature is intentionally small, but applications may add conservative primitives that are either client-local or shared-only, such as fixed projections, reshaping, model-specific differentiable maps, shared matrix multiplication, or shared linear solves. These extensions require uniform client-local realizations, agreement with ordinary evaluation on virtual global tensors, and no new federated-to-shared boundary. A formal statement is given as Proposition 7 in Appendix K.

5 Discussion

The introduced typed tensor language describes shared-state federated computation with virtual-global consistency. Local training can retain round-wise factorization while depending on the client partition. Exposure discipline gives no differential privacy or cryptographic confidentiality and does not bound inference from a shared statistic. Appendix E gives privacy lifting under its adjacency, sensitivity, and randomizer assumptions. Along these lines, future work will develop the privacy-preserving capabilities of our typed tensor language.

The learning results are expressibility results, not new optimizer-convergence theorems. When a represented federated update agrees with the corresponding centralized update on the virtual global tensor, any applicable centralized convergence theorem transfers under its own assumptions. The theory does not analyze statistical heterogeneity, client sampling, compression, stochastic approximation error, or optimizer-specific rates. Future work includes richer federated tensor languages and execution/backend optimizations for distributed tensor programs.

Acknowledgments and Disclosure of Funding

This project/research has received funding from the European Union's Research and Innovation Program Horizon Europe under Grant Agreement no. 101147319 (EBRAINS 2.0). We gratefully acknowledge NVIDIA Corporation for supporting this research with the donation of four NVIDIA RTX PRO 6000 Blackwell GPUs through the NVIDIA Academic Grant Program. We also thank PolyShape P.C., Greece, for covering the OpenAI usage costs incurred during this research.

References

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations (ICLR)*, 2021.

Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):23–34, 2013.

Jean-Philippe Bernardy and Patrik Jansson. Domain-specific tensor languages. *Journal of Functional Programming*, 35:e9, 2025.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, and Nicholas D. Lane Pedro Porto Buarque de GusmÃGo. Flower: a friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Akash Bharadwaj and Graham Cormode. An introduction to federated computation. In *International Conference on Management of Data (SIGMOD)*, pages 2448–2451, 2022.

Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. SystemDS: a declarative machine learning system for the end-to-end data science lifecycle. In *Conference on Innovative Data Systems Research (CIDR)*, 2020.

Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: system design. *Proceedings of Machine Learning and Systems (MLSys)*, 1:374–388, 2019.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

Rob Bonta. California consumer privacy act (ccpa). State of California Department of Justice, 2022. URL `https://oag.ca.gov/privacy/ccpa`.

Mark Braverman, Ankit Garg, Tengyu Ma, Huy L. Nguyen, and David P. Woodruff. Communication lower bounds for statistical estimation problems via a distributed data processing inequality. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. ACM, 2016. URL `https://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/bgmnw16.pdf`.

Graham Cormode and Daniel Ting. Federated data distribution shift estimation. *Proceedings of the VLDB Endowment*, 18(8):2399–2412, 2025.

Davide Domini, Gianluca Aguzzi, Lukas Esterle, and Mirko Viroli. FBFL: a field-based coordination approach for data heterogeneity in federated learning. *Logical Methods in Computer Science*, 22 (1):19:1–19:30, 2026. doi: 10.46298/lmcs-22(1:19)2026. URL `https://lmcs.episciences.org/17663`.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–487, 2014.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284, 2006.

David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.

Luciano Floridi and Mariarosaria Taddeo. What is data ethics? *Philosophical Transactions of the Royal Society A*, 374(2083), 2016.

Patrick Foley, Micah J. Sheller, Brandon Edwards, Sarthak Pati, Walter Riviera, Mansi Sharma, Prakash Narayana Moorthy, Shih han Wang, Jason Martin, Parsa Mirhaji, Prashant Shah, and Spyridon Bakas. OpenFL: the open federated learning library. *Physics in Medicine & Biology*, 67 (21):214001, 2022.

Yannis Foufoulas, Theofilos Mailis, Konstantinos Filippopolitis, Kalliopi-Christina Despotidou, Thanasis-Michail Karampatsis, Andreas Ktenidis, Evdokia Mailli, Theodore Papamarkou, and Yannis Ioannidis. Federated tensor algebra: A new programming paradigm for federated analytics and learning. In *Proceedings of the ACM AI Summit 2026 (AI Summit '26)*, Atlanta, GA, USA, 2026. Association for Computing Machinery. ISBN 979-8-4007-2638-5. doi: 10.1145/3806096.3844813.

Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to play any mental game, or a completeness theorem for protocols with honest majority*, pages 307–328. Association for Computing Machinery, 2019.

Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data Cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Programming Languages and Systems (ESOP 1998)*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi: 10.1007/BFb0053567.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 273–284. ACM, 2008. doi: 10.1145/1328897.1328472.

Peter Kairouz and H. Brendan McMahan. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *International Conference on Machine Learning (ICML)*, pages 5132–5143, 2020.

Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

Marcel Keller. MP-SPDZ: a versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590, 2020.

Sifis Lagouvardos, Julian Dolby, Neville Grech, Anastasios Antoniadis, and Yannis Smaragdakis. Static analysis of shape in TensorFlow programs. In *European Conference on Object-Oriented Programming (ECOOP)*, volume 166 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:29, 2020.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems (MLSys)*, 2:429–450, 2020.

Xiaoxiao Li, Meirui JIANG, Xiaofei Zhang, Michael Kamp, and Qi Dou. FedBN: federated learning on non-IID features via local batch normalization. In *International Conference on Learning Representations (ICLR)*, 2021.

Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated learning: privacy and incentive*, pages 240–254. Springer International Publishing, 2020.

Dougal Maclaurin, Alexey Radul, Matthew J. Johnson, and Dimitrios Vytiniotis. Dex: array programming with typed indices. In *Program Transformations for Machine Learning Workshop at NeurIPS 2019*, 2019.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282, 2017.

Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: a comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

Solmaz Niknam, Harpreet S. Dhillon, and Jeffrey H. Reed. Federated learning for wireless communications: motivation, opportunities, and challenges. *IEEE Communications Magazine*, 58(6):46–51, 2020.

Sharnil Pandya, Gautam Srivastava, Rutvij Jhaveri, M. Rajasekhara Babu, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Spyridon Mastorakis, Md. Jalil Piran, and Thippa Reddy Gadekallu. Federated learning for smart cities: a comprehensive survey. *Sustainable Energy Technologies and Assessments*, 55:102987, 2023.

Adam Paszke, Daniel Johnson, David Duvenaud, Dimitrios Vytiniotis, Alexey Radul, Matthew Johnson, Jonathan Ragan-Kelley, and Dougal Maclaurin. Getting to the point: Index sets and parallelism-preserving autodiff for pointful array programming. *Proceedings of the ACM on Programming Languages*, 5(ICFP), 2021. doi: 10.1145/3473593. URL `https://arxiv.org/abs/2104.05372`.

Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sébastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, and M. Jorge Cardoso. The future of digital health with federated learning. *NPJ Digital Medicine*, 3(1):119, 2020.

Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. NVIDIA FLARE: federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291*, 2022.

Holger R. Roth, Ziyue Xu, Yuan-Ting Hsieh, Adithya Renduchintala, Isaac Te-Chung Yang, Zhihong Zhang, Yuhong Wen, Sean Yang, Kevin Lu, Kristopher Kersten, Camir Ricketts, Daguang Xu, Chester Chen, Yan Cheng, and Andrew Feng. Empowering federated learning for massive models with NVIDIA FLARE. In *Federated learning systems: towards privacy-preserving distributed AI*, pages 1–17. Springer Nature Switzerland, 2025.

J Keith Rush, Zachary Charles, Zachary Garrett, Sean Augenstein, and Nicole Elyse Mitchell. DrJAX: scalable and differentiable MapReduce primitives in JAX. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024a.

Keith Rush, Zachary Charles, and Zachary Garrett. Federated automatic differentiation. *Journal of Machine Learning Research*, 25:1–39, 2024b.

TensorFlow. Tensorflow federated, 2021. URL `https://www.tensorflow.org/federated/get_started`. Official documentation for TensorFlow Federated and Federated Core; last updated 2021-04-22; accessed 2026-04-23.

The TensorFlow Federated Authors. TensorFlow Federated: machine learning on decentralized data, 2019. URL `https://www.tensorflow.org/federated`.

Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. From distributed coordination to field calculus and aggregate computing. *Journal of Logical and Algebraic Methods in Programming*, 109:100486, 2019. doi: 10.1016/j.jlamp.2019.100486.

Paul Voigt and Axel Von dem Bussche. *The EU general data protection regulation (GDPR): a practical guide*. Springer, 2017.

Philip Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, pages 273–286. ACM, 2012. doi: 10.1145/2364527.2364568.

Dan Wang, Siping Shi, Yifei Zhu, and Zhu Han. Federated analytics: opportunities and challenges. *IEEE Network*, 36(1):151–158, 2022.

Johannes Wehrstein, Tiemo Bang, Roman Heinrich, and Carsten Binnig. GRACEFUL: a learned cost estimator for UDFs. In *International Conference on Data Engineering (ICDE)*, pages 2450–2463, 2025.

Kaiqiang Xu, Di Chai, Junxue Zhang, Fan Lai, and Kai Chen. Sequoia: an accessible and extensible framework for privacy-preserving machine learning over distributed data. *Proceedings of the ACM on Management of Data*, 3(1):1–27, 2025.

Greg Yang. Wide feedforward or recurrent neural networks of any architecture are Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In *International Conference on Machine Learning (ICML)*, pages 11727–11737, 2021.

Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. Tensor relational algebra for distributed machine learning system design. *Proceedings of the VLDB Endowment*, 14(8):1338–1350, 2021.

Yuchen Zhang, John Duchi, Michael I. Jordan, and Martin J. Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, volume 26, 2013. URL `https://papers.nips.cc/paper_files/paper/2013/hash/d6ef5f7fa914c19931a55bb262ec879c-Abstract.html`.

Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, , Théo Ryffel, Zarreen Naowal Reza, and Georgios Kaissis. PySyft: a library for easy federated learning. In *Federated learning systems: towards next-generation AI*, pages 111–139. Springer International Publishing, 2021.

Shoshana Zuboff. The age of surveillance capitalism. In *Social theory re-wired*, pages 203–213. Routledge, 2023.

A Related work

This appendix gives an extended discussion of related work. The main paper focuses on the typed tensor language and its factorization theorems. Here we place the contribution in relation to federated learning and analytics, federated systems, declarative machine learning systems, tensor languages, differentiable distributed programming, and privacy mechanisms.

A.1 Federated learning, analytics, and computation

Federated learning provides the main context for this work. It studies model training from decentralized data while keeping raw records on client devices or within data-holding institutions [McMahan et al., 2017, Kairouz and McMahan, 2021]. The standard federated learning pattern is that clients compute local updates and a coordinating server aggregates those updates into a shared model. This pattern led to a large literature on optimization, systems, privacy, heterogeneity, and deployment [Bonawitz et al., 2019, Li et al., 2020]. Algorithms such as FedAvg [McMahan et al., 2017], FedProx [Li et al., 2020], and FedBN [Li et al., 2021] are important examples of this model-training view.

Federated analytics studies aggregate analysis over decentralized data without centralizing raw records [Wang et al., 2022]. Its goals overlap with federated learning but are not identical. Analytics tasks often compute summaries, histograms, distributional quantities, or other population-level statistics rather than model updates. Recent work on federated data distribution shift estimation gives one example of analytics-oriented federated computation beyond model training [Cormode and Ting, 2025]. More broadly, federated computation can be viewed as a larger class of distributed computations in which data stay local and only summary information is shared [Bharadwaj and Cormode, 2022].

Our work is closest to this broad federated computation view. The difference is that we do not begin from a particular deployed system, query interface, or optimizer. We define a typed tensor language and prove shared-state factorization results for the computations it expresses. This clarifies

Table 1: Comparison with TensorFlow Federated. The comparison is intended only to position the mathematical contribution, not to evaluate systems.

Aspect	TensorFlow Federated	Our typed tensor language
Primary role	Programming and execution framework for federated computations	Mathematical typed tensor language and semantic theory
Typing focus	Federated placements and structured values	Federated/shared tensor sorts, tracked record axis, and shared-state formation
Core semantics	Operational programming model for distributed computations	Virtual-global tensor semantics for correctness against centralized evaluation
Exposure boundary	Controlled by framework-level federated operators and placements	Formal exposure discipline through tensor types and record-axis elimination
Main theorem	Not primarily a theorem-driven formalization	Client-locality, virtual-global consistency, and shared-state factorization theorems

the distinction between client-local tensor computation, record-axis elimination into shared state, and shared-only post-processing.

Federated technologies are also motivated by privacy, governance, and institutional data-sharing constraints. These concerns arise in settings such as healthcare, finance, transportation, wireless networks, and the Internet of Things [Rieke et al., 2020, Long et al., 2020, Pandya et al., 2023, Niknam et al., 2020, Nguyen et al., 2021]. Legal and ethical discussions of data use, including GDPR and CCPA-oriented treatments, provide part of the broader context [Floridi and Taddeo, 2016, Zuboff, 2023, Voigt and Von dem Bussche, 2017, Bonta, 2022]. Our paper does not claim to solve legal compliance or privacy by itself. It gives an exposure discipline at the level of typed tensor semantics. Cryptographic, differential-privacy, or trusted-execution mechanisms remain separate layers.

A.2 Federated systems and TensorFlow Federated

Many federated learning systems provide APIs for building and deploying federated algorithms. TensorFlow Federated includes a Federated Core programming environment for distributed computations and a higher-level federated learning layer [TensorFlow, 2021, The TensorFlow Federated Authors, 2019]. Other systems, including OpenFL, Flower, PySyft, NVIDIA FLARE, and Sequoia, provide different abstractions for orchestration, client execution, secure computation, research experimentation, or distributed privacy-preserving workflows [Foley et al., 2022, Beutel et al., 2020, Ziller et al., 2021, Roth et al., 2022, 2025, Xu et al., 2025].

TensorFlow Federated (TFF) is the closest systems comparison because its Federated Core also provides a typed representation of federated computations. Our work is complementary rather than competing (see Table 1 for a comparison). TFF is an implementation framework for writing and executing federated computations. Our paper abstracts away runtime execution and gives tensor-level semantics that tracks the record axis, distinguishes federated tensors from shared tensors, and proves shared-state factorization theorems.

The distinction matters for the kind of results proved in our paper. A placement-aware framework can ensure that values are located at clients or at the server. Our type system refines this with tensor-level record-axis information. This lets us state, for example, that an expression treating a federated record axis as a shared tensor axis is invalid unless it passes through an explicit record-axis elimination. The result is a formal account of what the computation means and how it factors through shared state, rather than a proposal for how a specific runtime should schedule the computation.

Appendix L compares a covariance computation in FeTA's Python implementation with TFF orchestration of the same computation. FeTA infers the local and shared stages from operations on the data axis; both approaches retain the communication dependencies of the algorithm.

A.3 Declarative, algebraic, and database systems

A related line of work develops declarative or algebraic abstractions for machine learning systems. Tensor relational algebra gives an algebraic backend for distributed machine learning system design based on tensor relations and execution plans [Yuan et al., 2021]. SystemDS develops a declarative system for end-to-end data science pipelines and supports local, distributed, and federated execution [Boehm et al., 2020]. These works share with ours the view that structured mathematical computation should not be treated as a collection of opaque programs.

Declarative systems and algebraic backends are primarily concerned with specifying and optimizing computations across execution modes. Our paper is concerned with typed semantics for federated tensor computation. In particular, our language makes the transition from client-local state to shared state formal through the record axis. The shared-state factorization theorems then characterize computations that pass through fixed-dimensional shared state.

Classical database work on aggregation is also relevant. The data cube literature distinguishes distributive and algebraic aggregate functions from holistic aggregates requiring unbounded intermediate state [Gray et al., 1997]. Mergeable summaries give another formal account of summaries that can be combined from independently computed pieces [Agarwal et al., 2013]. Our shared-state factorizations are close in spirit to these ideas, but they are formulated for typed tensor expressions over federated records. They also connect the same finite-state principle to iterative learning updates, gradients, and curvature blocks.

Recent database work also studies user-defined functions from the perspective of query optimization. For example, GRACEFUL learns cost estimates for SQL query plans containing UDFs [Wehrstein et al., 2025]. This line is orthogonal to our focus. GRACEFUL studies execution-cost prediction for database query optimization, while our paper studies typed semantics and shared-state factorization for federated tensor computations.

A.4 Tensor languages, tensor programs, and differentiable distributed programming

Tensor and array languages study how to express structured tensor computation with semantic and typing principles. Dex uses typed indices for array programming [Maclaurin et al., 2019, Paszke et al., 2021], and static shape analysis for TensorFlow programs shows the value of reasoning about tensor shapes at the language level [Lagouvardos et al., 2020]. More broadly, recent work argues that tensor computation benefits from domain-specific language treatment rather than purely ad hoc library interfaces [Bernardy and Jansson, 2025].

Tensor programs are especially relevant as an example of a language accompanied by a theorem covering an expressible class. Tensor Programs I introduces a language for neural network computations and proves a Gaussian-process correspondence for networks expressible in that language [Yang, 2019]. Later tensor programs work develops the program further for feature learning and related infinite-width phenomena [Yang and Hu, 2021]. Our paper follows a similar high-level pattern in a different domain. We introduce a language for federated tensor computations and prove factorization theorems for the computations expressible in that language. Our theory is about when federated tensor computation can be carried out through fixed-dimensional shared state.

Federated automatic differentiation studies automatic differentiation for computations involving clients, a server, and communication between them [Rush et al., 2024b]. DrJAX develops differentiable MapReduce-style primitives in JAX for distributed and parallel machine learning [Rush et al., 2024a]. Our differentiable fragment assumes a user-supplied client-local expression for the per-sample gradient. It then proves that the empirical gradient and shared optimizer updates have shared-state factorizations. The theory does not construct this expression by automatic differentiation; the gradient assumption is part of Definition 8.

Field calculus gives local and aggregate semantics for distributed coordination [Viroli et al., 2019]. Field-based federated learning applies this programming abstraction to coordination under heterogeneous data [Domini et al., 2026]. Our focus is tensor expressions, sample-axis semantics, and shared-state factorizations.

Session types constrain communication protocols and the compatibility of their participants [Honda et al., 1998, 2008, Wadler, 2012]. Our types describe tensor placement and the data axis, from which

we prove locality, virtual-global consistency, and factorization for the program classes in Section 3. These results do not establish session-type properties for an execution backend.

Communication lower bounds for distributed statistical estimation concern the bits required to attain a statistical accuracy target under specified task and protocol assumptions [Braverman et al., 2016, Zhang et al., 2013]. The dimension $D(P)$ in Corollary 1 counts real coordinates in an extracted factorization. It gives a constructive upper bound on state size, without a bit bound or a minimum-state claim. Comparing this construction with those lower bounds requires a statistical task, an accuracy target, and a finite-precision model.

A.5 Privacy mechanisms and exposure discipline

The exposure discipline in this paper is a semantic statement about which tensors become shared. It should not be confused with a cryptographic or differential-privacy guarantee. Secure aggregation, secure multi-party computation, and differential privacy provide distinct privacy mechanisms [Bonawitz et al., 2017, Yao, 1982, Keller, 2020, Dwork et al., 2006]. These mechanisms can be used as backends or randomized layers for particular shared-state computations, but they are not implied by the deterministic type system.

At the same time, our typed tensor language gives a precise interface for privacy mechanisms. The shared-state factorization identifies the encoded local messages, the merged shared state, and the decoded output as distinct points where privacy mechanisms may be applied. Appendix E formalizes this observation through a privacy-lifting result for shared-state factorizations.

A.6 Summary of our distinct contribution

The closest existing work provides federated learning algorithms, analytics systems, distributed runtimes, declarative frameworks, tensor languages, differentiable distributed programming, or privacy backends. We define a typed tensor language for shared-state federated computation, prove semantic guarantees for its primitives, and characterize its one-round and iterative program classes through shared-state factorizations. The base class includes analytics, FedSGD, and shared server updates. The transient-state formulation in Appendix H permits multi-local-step training with a separate factorization guarantee; persistent private client state requires further semantics.

B Formal specification of the typed federated tensor language

B.1 Tensor types, contexts, and expressions

The semantic distinction of Subsection 2.1 is reflected at the type level. A shared type records an ordinary tensor shape. A federated type records a record axis together with the common non-record shape. This is required to accommodate federations with different client sample sizes. In what follows, $\mathcal{V}$ denotes a countably infinite set of variables.

Definition 9 (Tensor types)**.** *A tensor type is either a shared type* $\mathsf{Sh}(\mathbf{s})$ *or a federated type* $\mathsf{Fed}_r(\mathbf{d})$.

For a shared type, $\mathbf{s} = (s_1, \ldots, s_k)$ *is a tuple for some integer* $k \in \mathbb{N}_0$*, where each* s_j *is a positive integer when* $k \geq 1$*. The case* $k = 0$ *is identified with the empty tuple and corresponds to a scalar shared type.*

For a federated type, there exist an integer $k \geq 1$*, an axis* $r \in [k]$*, and positive integers* $d_1, \ldots, d_{r-1}, d_{r+1}, \ldots, d_k$ *such that* $\mathbf{d} = (d_1, \ldots, d_{r-1}, d_{r+1}, \ldots, d_k)$*. The type* $\mathsf{Fed}_r(\mathbf{d})$ *denotes a* k*-way federated tensor whose record axis is* r *and whose common non-record shape is* $\mathbf{d}$*. For* $k = 1$ *and* $r = 1$*, the tuple* $\mathbf{d}$ *is empty. Local sizes along axis* r *may be zero; all other extents and all shared extents are positive.*

Definition 10 (Typing context)**.** *A typing context is a finite partial function* Γ *from* $\mathcal{V}$ *to tensor types. If* $x \in \mathrm{dom}(\Gamma)$*, the value* $\Gamma(x)$ *is called the type of* x *under* Γ.

Definition 11 (Primitive signature)**.** *A primitive signature is a pair* $\Sigma = (\mathcal{F}, \mathrm{ar})$*, where* $\mathcal{F}$ *is a set and* $\mathrm{ar} : \mathcal{F} \to \mathbb{N}_0$ *is a function. If* $F \in \mathcal{F}$ *and* $\mathrm{ar}(F) = m$*, then* F *is called an* m*-ary primitive symbol.*

Definition 12 (Expressions)**.** *Let $\Sigma = (\mathcal{F}, \mathrm{ar})$ be a primitive signature. The set of expressions over Σ, denoted by $\mathrm{Expr}(\Sigma)$, is the smallest set satisfying the following conditions. If $x \in \mathcal{V}$, then $x \in \mathrm{Expr}(\Sigma)$. If $F \in \mathcal{F}$, if $\mathrm{ar}(F) = m$, and if $e_1, \ldots, e_m \in \mathrm{Expr}(\Sigma)$, then $F(e_1, \ldots, e_m) \in \mathrm{Expr}(\Sigma)$.*

Definition 13 (Typing judgment)**.** *Let Σ be a primitive signature. A typing judgment over Σ is an assertion of the form $\Gamma \vdash e : T$, where Γ is a typing context, $e \in \mathrm{Expr}(\Sigma)$, and T is a tensor type.*

Thus, syntax is defined independently of semantic interpretation. A primitive signature specifies only which primitive symbols may occur in expressions and how many inputs each symbol takes. Some typed primitive applications also require structural compatibility conditions that are not stored in the tensor type, such as equality of local record counts or local shapes across federated operands. These conditions are part of the definition of a valid typed primitive application in this specification.

B.2 Primitive signature

We now introduce the operator symbols of the base language. The present signature is deterministic. It contains element-wise maps, axis-wise aggregations, axis permutations, and three matrix-product symbols. Later sections enlarge this signature when first-order and second-order learning updates are studied.

Definition 14 (Aggregation schema)**.** *An aggregation schema is a family $\alpha = (\alpha_n)_{n \in \mathbb{N}_0}$, where $\alpha_n : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{R}^0 := \{()\}$. The value of α_0 is part of the data of α.*

Let $\mathcal{U}$, $\mathcal{B}$, $\mathcal{G}$, and $\mathcal{A}$ be pairwise disjoint collections with the following properties. Each $u \in \mathcal{U}$ is a map $u : \mathbb{R} \to \mathbb{R}$. Each $b \in \mathcal{B}$ is a map $b : \mathbb{R}^2 \to \mathbb{R}$. Each $g \in \mathcal{G}$ is a map $g : \mathbb{R}^2 \to \{0, 1\} \subseteq \mathbb{R}$. Each $\alpha \in \mathcal{A}$ is an aggregation schema in the sense of Definition 14. For each integer $k \geq 1$, let $\mathfrak{S}_k$ denote the permutation group of $[k]$, and define $\mathcal{P} := \bigcup_{k \geq 1} \mathfrak{S}_k$.

Definition 15 (Base primitive signature)**.** *The base primitive signature is the pair $\Sigma_0 = (\mathcal{F}_0, \mathrm{ar})$, where*

$$\mathcal{F}_0 := \mathcal{U} \cup \mathcal{B} \cup \mathcal{G} \cup \{\alpha_j : \alpha \in \mathcal{A},\ j \geq 1\} \cup \mathcal{P} \cup \{\mathsf{MatMul}_{\mathsf{FedSh}}, \mathsf{MatMul}_{\mathsf{ShFed}}, \mathsf{MatMul}_{\mathsf{FedFed}}\},$$

with all primitive symbols in the displayed union being regarded as formal tagged symbols, and the arity map ar is given by $\mathrm{ar}(u) = 1$, $\mathrm{ar}(b) = 2$, $\mathrm{ar}(g) = 2$, $\mathrm{ar}(\alpha_j) = 1$, and $\mathrm{ar}(\tau) = 1$ for every $u \in \mathcal{U}$, $b \in \mathcal{B}$, $g \in \mathcal{G}$, $\alpha \in \mathcal{A}$, integer $j \geq 1$, and $\tau \in \mathcal{P}$, together with $\mathrm{ar}(\mathsf{MatMul}_{\mathsf{FedSh}}) = \mathrm{ar}(\mathsf{MatMul}_{\mathsf{ShFed}}) = \mathrm{ar}(\mathsf{MatMul}_{\mathsf{FedFed}}) = 2$.

We use the same notation for a primitive symbol and for its later tensor interpretation. Symbols in $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{G}$ will be interpreted element-wise. Symbols of the form α_j will be interpreted as aggregation along axis j. A permutation $\tau \in \mathfrak{S}_k$ will be interpreted as reordering the axes of a k-way tensor according to τ. The three matrix-product symbols distinguish the federated-shared, shared-federated, and federated-federated cases.

The signature records only operator symbols and their arities, without imposing shape compatibility, broadcasting rules, or record-axis constraints. Those conditions enter through the typing rules and the semantic clauses.

B.3 Ordinary tensor operations induced by the signature

We first interpret the non-matrix primitive symbols on ordinary tensors. These interpretations will later be lifted to shared tensors and federated tensors. Matrix-product symbols are treated separately because their meaning depends on tensor kind and on the position of the record axis.

Definition 16 (Index sets and axis deletion)**.** *Let $\mathbf{s} = (s_1, \ldots, s_k)$ be a shape tuple with $k \in \mathbb{N}_0$. Define the index set of $\mathbf{s}$ by*

$$I(\mathbf{s}) = \begin{cases} [s_1] \times \ldots \times [s_k], & k \geq 1, \\ \{()\}, & k = 0. \end{cases}$$

If $k \geq 1$ and $j \in [k]$, define $\mathbf{s} \setminus j = (s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_k)$.

Definition 17 (Broadcast compatibility)**.** *Let $\mathbf{s} = (s_1, \ldots, s_p)$ and $\mathbf{t} = (t_1, \ldots, t_q)$ be shape tuples with $p, q \in \mathbb{N}_0$. Set $\ell := \max\{p, q\}$. Define the left-padded tuples $\mathrm{pad}_\ell(\mathbf{s}) = (\tilde{s}_1, \ldots, \tilde{s}_\ell)$ and*

$\mathrm{pad}_\ell(\mathbf{t}) = (\tilde{t}_1, \ldots, \tilde{t}_\ell)$ *by*

$$\tilde{s}_i = \begin{cases} 1, & 1 \le i \le \ell - p, \\ s_{i-\ell+p}, & \ell - p + 1 \le i \le \ell, \end{cases} \qquad \tilde{t}_i = \begin{cases} 1, & 1 \le i \le \ell - q, \\ t_{i-\ell+q}, & \ell - q + 1 \le i \le \ell. \end{cases}$$

The tuples $\mathbf{s}$ *and* $\mathbf{t}$ *are broadcast-compatible if, for every* $i \in [\ell]$*, at least one of the following holds:* $\tilde{s}_i = \tilde{t}_i$*,* $\tilde{s}_i = 1$*, or* $\tilde{t}_i = 1$*. If* $\mathbf{s}$ *and* $\mathbf{t}$ *are broadcast-compatible, their broadcast shape is* $\mathbf{s} \vee \mathbf{t} = (u_1, \ldots, u_\ell)$*. For* $i \in [\ell]$*, set*

$$u_i := \begin{cases} \tilde{t}_i, & \tilde{s}_i = 1, \\ \tilde{s}_i, & \tilde{s}_i \neq 1. \end{cases}$$

Thus, broadcasting an extent of zero against one retains zero.

Definition 18 (Broadcast map)**.** *Let* $\mathbf{s}$ *and* $\mathbf{u}$ *be shape tuples such that* $\mathbf{s}$ *is broadcast-compatible with* $\mathbf{u}$ *and such that* $\mathbf{s} \vee \mathbf{u} = \mathbf{u}$*. Let* $\mathbf{s} = (s_1, \ldots, s_p)$ *and* $\mathbf{u} = (u_1, \ldots, u_\ell)$*, with* $p, \ell \in \mathbb{N}_0$ *and* $p \le \ell$*. Write* $\mathrm{pad}_\ell(\mathbf{s}) = (\tilde{s}_1, \ldots, \tilde{s}_\ell)$*. Define* $\pi^{\mathbf{u}}_{\mathbf{s}} : I(\mathbf{u}) \to I(\mathbf{s})$ *as follows. If* $p = 0$*, then* $\pi^{\mathbf{u}}_{\mathbf{s}}(i) = ()$ *for all* $i \in I(\mathbf{u})$*. If* $p \ge 1$ *and* $i = (i_1, \ldots, i_\ell) \in I(\mathbf{u})$*, then* $\pi^{\mathbf{u}}_{\mathbf{s}}(i) = (j_1, \ldots, j_p)$*, where for each* $q \in [p]$*,*

$$j_q = \begin{cases} 1, & s_q = 1, \\ i_{\ell-p+q}, & s_q \neq 1. \end{cases}$$

Define the broadcast map $\mathrm{Br}^{\mathbf{u}}_{\mathbf{s}} : \mathbb{R}^{\mathbf{s}} \to \mathbb{R}^{\mathbf{u}}$ *as follows. If* $T \in \mathbb{R}^{\mathbf{s}}$*, then* $\mathrm{Br}^{\mathbf{u}}_{\mathbf{s}}(T) \in \mathbb{R}^{\mathbf{u}}$ *is defined by* $\mathrm{Br}^{\mathbf{u}}_{\mathbf{s}}(T)[i] = T\left[\pi^{\mathbf{u}}_{\mathbf{s}}(i)\right]$ *for every* $i \in I(\mathbf{u})$*.*

Definition 19 (Ordinary tensor operations induced by the signature)**.** *Let* Σ_0 *be the base primitive signature of Subsection B.2.*

If $u \in \mathcal{U}$ *and* $T \in \mathbb{R}^{\mathbf{s}}$*, define* $u(T) \in \mathbb{R}^{\mathbf{s}}$ *by* $u(T)[i] = u(T[i])$ *for every* $i \in I(\mathbf{s})$*.*

If $b \in \mathcal{B}$*,* $S \in \mathbb{R}^{\mathbf{s}}$*, and* $T \in \mathbb{R}^{\mathbf{t}}$*, assume that* $\mathbf{s}$ *and* $\mathbf{t}$ *are broadcast-compatible and set* $\mathbf{u} := \mathbf{s} \vee \mathbf{t}$*. Define* $b(S, T) \in \mathbb{R}^{\mathbf{u}}$ *by* $b(S, T)[i] = b\left(\mathrm{Br}^{\mathbf{u}}_{\mathbf{s}}(S)[i], \mathrm{Br}^{\mathbf{u}}_{\mathbf{t}}(T)[i]\right)$ *for every* $i \in I(\mathbf{u})$*.*

If $g \in \mathcal{G}$*,* $S \in \mathbb{R}^{\mathbf{s}}$*, and* $T \in \mathbb{R}^{\mathbf{t}}$*, assume that* $\mathbf{s}$ *and* $\mathbf{t}$ *are broadcast-compatible and set* $\mathbf{u} := \mathbf{s} \vee \mathbf{t}$*. Define* $g(S, T) \in \mathbb{R}^{\mathbf{u}}$ *by* $g(S, T)[i] = g\left(\mathrm{Br}^{\mathbf{u}}_{\mathbf{s}}(S)[i], \mathrm{Br}^{\mathbf{u}}_{\mathbf{t}}(T)[i]\right)$ *for every* $i \in I(\mathbf{u})$*.*

If $\alpha \in \mathcal{A}$*,* $k \ge 1$*,* $T \in \mathbb{R}^{s_1 \times \cdots \times s_k}$*, and* $j \in [k]$*, define* $\alpha_j(T) \in \mathbb{R}^{\mathbf{s} \setminus j}$*, where* $\mathbf{s} := (s_1, \ldots, s_k)$*, by the rule*

$$\alpha_j(T)[i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k] = \\ \alpha_{s_j}\left(T[i_1, \ldots, i_{j-1}, 1, i_{j+1}, \ldots, i_k], \ldots, T[i_1, \ldots, i_{j-1}, s_j, i_{j+1}, \ldots, i_k]\right)$$

for every $(i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k) \in I(\mathbf{s} \setminus j)$*. If* $k = 1$ *and* $j = 1$*, then* $\alpha_1(T) \in \mathbb{R}$ *is given by* $\alpha_1(T) = \alpha_{s_1}(T[1], \ldots, T[s_1])$*.*

If $\tau \in \mathfrak{S}_k$ *and* $T \in \mathbb{R}^{s_1 \times \cdots \times s_k}$*, define* $\tau \cdot (s_1, \ldots, s_k) = (s_{\tau^{-1}(1)}, \ldots, s_{\tau^{-1}(k)})$*, and define* $\tau(T) \in \mathbb{R}^{\tau \cdot (s_1, \ldots, s_k)}$ *by* $\tau(T)[i_1, \ldots, i_k] = T[i_{\tau(1)}, \ldots, i_{\tau(k)}]$ *for every* $(i_1, \ldots, i_k) \in I(\tau \cdot (s_1, \ldots, s_k))$*.*

Definition 19 gives the ordinary tensor meaning of unary maps, binary element-wise maps, aggregation symbols, and axis permutations. These ordinary operations will be combined with the two tensor kinds of Subsection 2.1 in the next subsection.

B.4 Semantics of non-matrix primitives on shared and federated tensors

We interpret the non-matrix symbols of Σ_0 on shared tensors and federated tensors. Matrix-product symbols are treated separately because their output kind depends on how the record axis is used.

Definition 20 (Ordinary shape)**.** *Let* T *be an ordinary tensor or a scalar. There exists a unique shape tuple* $\mathbf{s}$ *such that* $T \in \mathbb{R}^{\mathbf{s}}$*. Define the ordinary shape of* T *as* $\mathrm{shape}(T) := \mathbf{s}$*.*

Definition 21 (Local broadcast of a shared tensor against a federated tensor)**.** *Let* $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ *be a federated tensor. Let* S *be a shared tensor. Assume that, for every* $c \in \mathcal{C}$*, the tuples* $\mathrm{shape}(S)$ *and* $\mathrm{shape}(X^{(c)})$ *are broadcast-compatible and satisfy* $\mathrm{shape}(S) \vee \mathrm{shape}(X^{(c)}) = \mathrm{shape}(X^{(c)})$*. For each* $c \in \mathcal{C}$*, define the local broadcast of* S *against* X *as* $\mathrm{Br}_c(S; X) := \mathrm{Br}^{\mathrm{shape}(X^{(c)})}_{\mathrm{shape}(S)}(S) \in \mathbb{R}^{\mathrm{shape}(X^{(c)})}$*.*

Definition 22 (Unary semantics)**.** *Let $u \in \mathcal{U}$. If S is a shared tensor, define $u(S)$ to be the shared tensor obtained by applying the ordinary tensor operation u of Definition 19 to S. If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor with record axis r, define the unary semantics of X as $u(X) := \{u(X^{(c)})\}_{c\in\mathcal{C}}$.*

Definition 23 (Binary element-wise semantics)**.** *Let $\beta \in \mathcal{B} \cup \mathcal{G}$.*

If S and T are shared tensors and $\mathrm{shape}(S)$ and $\mathrm{shape}(T)$ are broadcast-compatible, define the binary element-wise semantics $\beta(S,T)$ of S and T as the shared tensor obtained by applying the ordinary tensor operation β of Definition 19 to S and T.

If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ and $Z = \{Z^{(c)}\}_{c\in\mathcal{C}}$ are federated tensors with the same record axis r and satisfying $\mathrm{shape}(X^{(c)}) = \mathrm{shape}(Z^{(c)})$ for every $c \in \mathcal{C}$, define $\beta(X,Z) := \{\beta(X^{(c)}, Z^{(c)})\}_{c\in\mathcal{C}}$.

If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor with record axis r, if S is a shared tensor, and if $\mathrm{Br}_c(S;X)$ is defined for every $c \in \mathcal{C}$, define $\beta(X,S) := \{\beta(X^{(c)}, \mathrm{Br}_c(S;X))\}_{c\in\mathcal{C}}$.

If S is a shared tensor, if $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor with record axis r, and if $\mathrm{Br}_c(S;X)$ is defined for every $c \in \mathcal{C}$, define $\beta(S,X) := \{\beta(\mathrm{Br}_c(S;X), X^{(c)})\}_{c\in\mathcal{C}}$.

Definition 24 (Aggregation semantics)**.** *Let $\alpha \in \mathcal{A}$.*

If S is a shared tensor of ordinary order $k \geq 1$ and if $j \in [k]$, define $\alpha_j(S)$ to be the shared tensor obtained by applying the ordinary tensor operation α_j of Definition 19 to S.

If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor of ordinary order $k \geq 1$ with record axis r and if $j = r$, define the aggregation semantics of X along axis r as $\alpha_r(X) := \alpha_r(\overline{X})$.

If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor of ordinary order $k \geq 1$ with record axis r and if $j \in [k]\setminus\{r\}$, define $\alpha_j(X) := \{\alpha_j(X^{(c)})\}_{c\in\mathcal{C}}$. Then $\alpha_j(X)$ is a federated tensor with record axis

$$r' = \begin{cases} r, & j > r, \\ r-1, & j < r. \end{cases}$$

Definition 25 (Permutation semantics)**.** *Let $k \geq 1$ and let $\tau \in \mathfrak{S}_k$.*

If S is a shared tensor of ordinary order k, define the permutation semantics $\tau(S)$ of S to be the shared tensor obtained by applying the ordinary tensor operation τ of Definition 19 to S.

If $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ is a federated tensor of ordinary order k with record axis r, define $\tau(X) := \{\tau(X^{(c)})\}_{c\in\mathcal{C}}$.

It follows from Definitions 22, 23, 24, and 25 that $u(X)$ is a federated tensor with record axis r, each of $\beta(X,Z)$, $\beta(X,S)$, and $\beta(S,X)$ is a federated tensor with record axis r, $\alpha_r(X)$ is a shared tensor, and $\tau(X)$ is a federated tensor with record axis $\tau(r)$. These semantic definitions specify how non-matrix primitives act on shared tensors and federated tensors. They also identify the only non-matrix operation that can produce shared state from federated inputs, namely aggregation along the record axis.

B.5 Semantics of matrix-product primitives

We now interpret the three matrix-product symbols in Σ_0. Each symbol is partial. It is defined for inputs whose matrix shapes and record-axis positions satisfy the stated conditions. These are the cases needed to preserve the federated kind or to eliminate the record axis into shared state.

A shared matrix is a shared tensor of ordinary order 2. A federated matrix is a federated tensor of ordinary order 2.

Definition 26 (Federated-shared matrix product)**.** *Let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ be a federated matrix with record axis 1. Assume that there exists an integer $p \geq 1$ and a family of local record counts $\{n_c\}_{c\in\mathcal{C}} \subseteq \mathbb{N}_0$ such that $X^{(c)} \in \mathbb{R}^{n_c\times p}$ for every $c \in \mathcal{C}$. Let $S \in \mathbb{R}^{p\times q}$ be a shared matrix for some integer $q \geq 1$. Define the federated-shared matrix product as $\mathsf{MatMul}_{\mathsf{FedSh}}(X,S) := \{X^{(c)}S\}_{c\in\mathcal{C}}$.*

Definition 27 (Shared-federated matrix product)**.** *Let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ be a federated matrix with record axis 2. Assume that there exists an integer $p \geq 1$ and a family of local record counts $\{n_c\}_{c\in\mathcal{C}} \subseteq \mathbb{N}_0$ such that $X^{(c)} \in \mathbb{R}^{p\times n_c}$ for every $c \in \mathcal{C}$. Let $S \in \mathbb{R}^{q\times p}$ be a shared matrix for some integer $q \geq 1$. Define the shared-federated matrix product as $\mathsf{MatMul}_{\mathsf{ShFed}}(S,X) := \{SX^{(c)}\}_{c\in\mathcal{C}}$.*

Definition 28 (Federated-federated matrix product)**.** *Let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ and $Z = \{Z^{(c)}\}_{c\in\mathcal{C}}$ be federated matrices. Assume that X has record axis 2, that Z has record axis 1, and that there exist integers $a, b \geq 1$ and a family of local record counts $\{n_c\}_{c\in\mathcal{C}} \subseteq \mathbb{N}_0$ such that $X^{(c)} \in \mathbb{R}^{a\times n_c}$ and $Z^{(c)} \in \mathbb{R}^{n_c\times b}$ for every $c \in \mathcal{C}$. Define the federated-federated matrix product as $\mathsf{MatMul}_{\mathsf{FedFed}}(X, Z) := \sum_{c\in\mathcal{C}} X^{(c)}Z^{(c)}$.*

It follows from Definitions 26, 27, and 28 that $\mathsf{MatMul}_{\mathsf{FedSh}}(X, S)$ is a federated matrix with record axis 1 and local shapes $X^{(c)}S \in \mathbb{R}^{n_c\times q}$ for every $c \in \mathcal{C}$, $\mathsf{MatMul}_{\mathsf{ShFed}}(S, X)$ is a federated matrix with record axis 2 and local shapes $SX^{(c)} \in \mathbb{R}^{q\times n_c}$ for every $c \in \mathcal{C}$, and $\mathsf{MatMul}_{\mathsf{FedFed}}(X, Z)$ is a shared matrix in $\mathbb{R}^{a\times b}$.

The federated-shared and shared-federated matrix products preserve the federated structure. The federated-federated matrix product eliminates the record axis and produces shared state. Other axis placements can be reduced to these cases by composing with permutation primitives. Shared-only matrix multiplication, which is not part of the base signature, will be added when later sections expand the language for shared-only post-processing and second-order updates.

B.6 Type-level shape operations

We introduce the type-level operations needed to state the typing rules. The main issue is that a federated type records a record axis but does not record local record counts. For that reason, type-level broadcasting against a federated type must be defined without reference to any client sample size.

Definition 29 (Symbolic shape of a federated type)**.** *Let $T = \mathsf{Fed}_r(\mathbf{d})$, $\mathbf{d} = (d_1, \ldots, d_{k-1})$, where $k \geq 1$ and $r \in [k]$. In this definition, the $k-1$ entries of $\mathbf{d}$ are indexed consecutively, preserving their order after axis r is omitted. Define the symbolic shape of T as the tuple $\mathrm{sym}(T) = (\sigma_1, \ldots, \sigma_k)$ with entries in $\mathbb{N} \cup \{\star\}$ given by*

$$\sigma_j = \begin{cases} d_j, & 1 \leq j < r, \\ \star, & j = r, \\ d_{j-1}, & r < j \leq k. \end{cases}$$

The tuple $\mathrm{sym}(T)$ of Definition 29 has exactly one coordinate equal to $\star$. That coordinate records the record axis.

Definition 30 (Erasure of the record marker)**.** *Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$ be a tuple with entries in $\mathbb{N}\cup\{\star\}$ and with exactly one coordinate equal to $\star$. Define the erasure $\mathrm{erase}_\star(\boldsymbol{\sigma})$ of the record marker in $\boldsymbol{\sigma}$ as the tuple of positive integers obtained by deleting the unique coordinate equal to $\star$ and preserving the order of the remaining coordinates.*

Definition 31 (Record-agnostic broadcast compatibility)**.** *Let $\mathbf{s} = (s_1, \ldots, s_p)$ be a shared shape tuple with $p \in \mathbb{N}_0$. Let $T = \mathsf{Fed}_r(\mathbf{d})$ be a federated type whose symbolic shape is $\mathrm{sym}(T) = (\sigma_1, \ldots, \sigma_k)$. If $p > k$, then $\mathbf{s}$ is not record-agnostically broadcast-compatible with T. Assume henceforth that $p \leq k$. Define the left-padded tuple $\mathrm{pad}_k(\mathbf{s}) = (\tilde{s}_1, \ldots, \tilde{s}_k)$ by padding $\mathbf{s}$ on the left with entries equal to 1.*

We say that $\mathbf{s}$ is record-agnostically broadcast-compatible with T if, for every $i \in [k]$, the following conditions hold. If $\sigma_i = \star$, then $\tilde{s}_i = 1$. If $\sigma_i \in \mathbb{N}$, then $\tilde{s}_i \in \{1, \sigma_i\}$.

The previous condition depends only on the type $\mathsf{Fed}_r(\mathbf{d})$, without a dependence on any local record count. It is the static condition that guarantees a shared tensor of shape $\mathbf{s}$ can be broadcast against every local tensor having federated type T, without changing the local tensor shape.

Definition 32 (Deletion of a non-record axis from a federated type)**.** *Let $T = \mathsf{Fed}_r(\mathbf{d})$, $\mathrm{sym}(T) = (\sigma_1, \ldots, \sigma_k)$, with $k \geq 1$. Let $j \in [k] \setminus \{r\}$. Define the deletion of axis j from T as follows. Let $\boldsymbol{\sigma}' = (\sigma_1, \ldots, \sigma_{j-1}, \sigma_{j+1}, \ldots, \sigma_k)$. Let r' be the unique index in $[k-1]$ such that the r'-th coordinate of $\boldsymbol{\sigma}'$ is equal to $\star$. Let $\mathbf{d}' := \mathrm{erase}_\star(\boldsymbol{\sigma}')$ as in Definition 30. Define $\mathrm{del}_j(T) := \mathsf{Fed}_{r'}(\mathbf{d}')$.*

Definition 33 (Permutation of a federated type)**.** *Let $T = \mathsf{Fed}_r(\mathbf{d})$, $\mathrm{sym}(T) = (\sigma_1, \ldots, \sigma_k)$, with $k \geq 1$, and let $\tau \in \mathfrak{S}_k$. Define the permutation of T by τ as follows. Define $\tau \cdot \mathrm{sym}(T) = (\sigma_{\tau^{-1}(1)}, \ldots, \sigma_{\tau^{-1}(k)})$. Let $\mathbf{d}' := \mathrm{erase}_\star(\tau \cdot \mathrm{sym}(T))$ as in Definition 30. Define $\tau(T) := \mathsf{Fed}_{\tau(r)}(\mathbf{d}')$.*

B.7 Typing rules for non-matrix primitives

In this subsection, all expressions are expressions over the base primitive signature Σ_0. The typing rules below assign output types to primitive applications whenever the corresponding semantic compatibility conditions are satisfied. For every integer $k \geq 1$, every permutation $\tau \in \mathfrak{S}_k$, and every shape tuple $\mathbf{s} = (s_1, \ldots, s_k)$, we write $\tau \cdot \mathbf{s} := (s_{\tau^{-1}(1)}, \ldots, s_{\tau^{-1}(k)})$.

Definition 34 (Typing of unary primitives)**.** *Let $u \in \mathcal{U}$.*

If $\Gamma \vdash e : \mathsf{Sh}(\mathbf{s})$, then define the typing rule for u on e by $\Gamma \vdash u(e) : \mathsf{Sh}(\mathbf{s})$.

If $\Gamma \vdash e : \mathsf{Fed}_r(\mathbf{d})$, then define the typing rule for u on e by $\Gamma \vdash u(e) : \mathsf{Fed}_r(\mathbf{d})$.

Definition 35 (Typing of binary element-wise primitives)**.** *Let $\beta \in \mathcal{B} \cup \mathcal{G}$.*

If $\Gamma \vdash e : \mathsf{Sh}(\mathbf{s})$ and $\Gamma \vdash f : \mathsf{Sh}(\mathbf{t})$, and if $\mathbf{s}$ and $\mathbf{t}$ are broadcast-compatible, then define the typing rule for β on (e, f) by $\Gamma \vdash \beta(e, f) : \mathsf{Sh}(\mathbf{s} \vee \mathbf{t})$.

If $\Gamma \vdash e : \mathsf{Fed}_r(\mathbf{d})$ and $\Gamma \vdash f : \mathsf{Fed}_r(\mathbf{d})$, and if the corresponding federated primitive application satisfies the local-shape compatibility condition of Definition 23, then define the typing rule for β on (e, f) by $\Gamma \vdash \beta(e, f) : \mathsf{Fed}_r(\mathbf{d})$.

If $\Gamma \vdash e : \mathsf{Fed}_r(\mathbf{d})$ and $\Gamma \vdash f : \mathsf{Sh}(\mathbf{s})$, and if $\mathbf{s}$ is record-agnostically broadcast-compatible with $\mathsf{Fed}_r(\mathbf{d})$ according to Definition 31, then define the typing rule for β on (e, f) by $\Gamma \vdash \beta(e, f) : \mathsf{Fed}_r(\mathbf{d})$.

If $\Gamma \vdash e : \mathsf{Sh}(\mathbf{s})$ and $\Gamma \vdash f : \mathsf{Fed}_r(\mathbf{d})$, and if $\mathbf{s}$ is record-agnostically broadcast-compatible with $\mathsf{Fed}_r(\mathbf{d})$ according to Definition 31, then define the typing rule for β on (e, f) by $\Gamma \vdash \beta(e, f) : \mathsf{Fed}_r(\mathbf{d})$.

Definition 36 (Typing of aggregation primitives)**.** *Let $\alpha \in \mathcal{A}$. Let $\mathbf{s} = (s_1, \ldots, s_k)$ with $k \geq 1$. If $\Gamma \vdash e : \mathsf{Sh}(\mathbf{s})$ and $j \in [k]$, then define the typing rule for α_j on e by $\Gamma \vdash \alpha_j(e) : \mathsf{Sh}(\mathbf{s} \setminus j)$.*

Let $T = \mathsf{Fed}_r(\mathbf{d})$, where $r \in [k]$ and $k = |\mathbf{d}| + 1$. If $\Gamma \vdash e : T$, then define the typing rule for record-axis aggregation on e by $\Gamma \vdash \alpha_r(e) : \mathsf{Sh}(\mathbf{d})$.

If $\Gamma \vdash e : T$ and $j \in [k] \setminus \{r\}$, then define the typing rule for non-record-axis aggregation on e by $\Gamma \vdash \alpha_j(e) : \mathrm{del}_j(T)$, where $\mathrm{del}_j(T)$ is the federated type of Definition 32.

Definition 37 (Typing of permutation primitives)**.** *Let $k \geq 1$ and let $\tau \in \mathfrak{S}_k$.*

If $\Gamma \vdash e : \mathsf{Sh}(\mathbf{s})$, where $\mathbf{s} = (s_1, \ldots, s_k)$, then define the typing rule for τ on e by $\Gamma \vdash \tau(e) : \mathsf{Sh}(\tau \cdot \mathbf{s})$.

If $\Gamma \vdash e : \mathsf{Fed}_r(\mathbf{d})$ and $k = |\mathbf{d}| + 1$, then define the typing rule for τ on e by $\Gamma \vdash \tau(e) : \tau(\mathsf{Fed}_r(\mathbf{d}))$, where $\tau(\mathsf{Fed}_r(\mathbf{d}))$ is the federated type of Definition 33.

The rules of Definitions 34–37 provide the two ways in which a primitive may act on federated data. It may preserve federated structure, as in unary maps, element-wise operations, non-record-axis aggregation, and permutations. It may also eliminate the record axis and produce shared state, as in record-axis aggregation. The remaining shared-output primitive in the base language is the federated-federated matrix product, which is typed separately.

B.8 Typing rules for matrix-product primitives

We now state the typing rules for the three matrix-product primitives in the base language. These are the only matrix products in Σ_0. They are chosen so that matrix multiplication either preserves the federated structure or eliminates the record axis into shared state.

Definition 38 (Typing of the federated-shared matrix product)**.** *Let $p, q \geq 1$. If $\Gamma \vdash e : \mathsf{Fed}_1((p))$ and $\Gamma \vdash f : \mathsf{Sh}((p, q))$, then define the typing rule for the federated-shared matrix product by $\Gamma \vdash \mathsf{MatMul}_{\mathsf{FedSh}}(e, f) : \mathsf{Fed}_1((q))$.*

Definition 39 (Typing of the shared-federated matrix product)**.** *Let $p, q \geq 1$. If $\Gamma \vdash e : \mathsf{Sh}((q, p))$ and $\Gamma \vdash f : \mathsf{Fed}_2((p))$, then define the typing rule for the shared-federated matrix product by $\Gamma \vdash \mathsf{MatMul}_{\mathsf{ShFed}}(e, f) : \mathsf{Fed}_2((q))$.*

Definition 40 (Typing of the federated-federated matrix product)**.** *Let $a, b \geq 1$. If $\Gamma \vdash e : \mathsf{Fed}_2((a))$ and $\Gamma \vdash f : \mathsf{Fed}_1((b))$, and if the corresponding primitive application satisfies the client-wise local-count compatibility condition of Definition 28, then define the typing rule for the federated-federated matrix product by $\Gamma \vdash \mathsf{MatMul}_{\mathsf{FedFed}}(e, f) : \mathsf{Sh}((a, b))$.*

The typing rules of Definitions 38 and 39 preserve the federated kind. The typing rule of Definition 40 produces a shared tensor. Other placements of the record axis can be reduced to these cases by combining matrix-product primitives with permutation primitives. Because federated types do not record local record counts, the federated-federated matrix product rule carries a semantic side condition. It applies only when the locally contracted dimensions agree on every client.

C Proofs of foundational semantic propositions

Proof of Proposition 1. Fix a primitive symbol F of Σ_0. Consider a typed primitive application of F whose federated inputs are $X_1, \dots, X_p$ and whose shared inputs are $S_1, \dots, S_q$, and let Y denote its output. Assume that Y is a federated tensor. We show that, for each client $c \in \mathcal{C}$, the local output $Y^{(c)}$ depends only on the local federated inputs at c and on the shared inputs.

We proceed by a case analysis over the federated-output primitives of the base language.

Assume first that $F = u$ for some $u \in \mathcal{U}$. Then there is one federated input and no shared input. By Definition 22, $Y^{(c)} = u(X_1^{(c)})$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A) := u(A)$.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$ and that both inputs are federated. Then there are two federated inputs and no shared input. By Definition 23, $Y^{(c)} = \beta(X_1^{(c)}, X_2^{(c)})$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A, B) := \beta(A, B)$.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$, that there is one federated input X_1, and that there is one shared input S_1 used in the second operand position. By Definition 23, $Y^{(c)} = \beta(X_1^{(c)}, \mathrm{Br}_c(S_1; X_1))$ for every $c \in \mathcal{C}$. For fixed c, define $\Phi_{F,c}(A, S) := \beta(A, \mathrm{Br}^{\mathrm{shape}(A)}_{\mathrm{shape}(S)}(S))$ for tensors A and S of the same shapes as $X_1^{(c)}$ and S_1, respectively (see Definitions 20 and 18). Substituting $A = X_1^{(c)}$ and $S = S_1$ gives $Y^{(c)} = \Phi_{F,c}(X_1^{(c)}, S_1)$.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$, that there is one shared input S_1 used in the first operand position, and that there is one federated input X_1. By Definition 23, $Y^{(c)} = \beta(\mathrm{Br}_c(S_1; X_1), X_1^{(c)})$ for every $c \in \mathcal{C}$. For fixed c, define $\Phi_{F,c}(A, S) := \beta(\mathrm{Br}^{\mathrm{shape}(A)}_{\mathrm{shape}(S)}(S), A)$ for tensors A and S of the same shapes as $X_1^{(c)}$ and S_1, respectively. Substituting $A = X_1^{(c)}$ and $S = S_1$ gives $Y^{(c)} = \Phi_{F,c}(X_1^{(c)}, S_1)$.

Assume next that $F = \alpha_j$ for some $\alpha \in \mathcal{A}$ and some axis j. Since the output is federated, Definition 36 implies that the input is a federated tensor X_1 and that j is not its record axis. By Definition 24, $Y^{(c)} = \alpha_j(X_1^{(c)})$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A) := \alpha_j(A)$.

Assume next that $F = \tau$ for some permutation $\tau \in \mathfrak{S}_k$. Since the output is federated, the input is a federated tensor X_1 of ordinary order k by Definition 37. By Definition 25, $Y^{(c)} = \tau(X_1^{(c)})$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A) := \tau(A)$.

Assume next that $F = \mathsf{MatMul}_{\mathsf{FedSh}}$. Then there is one federated input X_1 and one shared input S_1. By Definition 26, $Y^{(c)} = X_1^{(c)} S_1$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A, S) := AS$.

Assume finally that $F = \mathsf{MatMul}_{\mathsf{ShFed}}$. Then there is one shared input S_1 and one federated input X_1. By Definition 27, $Y^{(c)} = S_1 X_1^{(c)}$ for every $c \in \mathcal{C}$. Hence the required map is $\Phi_{F,c}(A, S) := SA$.

These cases exhaust the federated-output primitives of the base language by Definitions 22, 23, 24, 25, 26, and 27. In each case, the constructed map is determined by the primitive symbol, the operand positions, and the local input shapes. It does not depend on the identity of c or on any tensor stored at another client. Therefore, for every client $c \in \mathcal{C}$, there exists a deterministic map $\Phi_{F,c}$ such that $Y^{(c)} = \Phi_{F,c}(X_1^{(c)}, \dots, X_p^{(c)}, S_1, \dots, S_q)$. This proves the proposition. □

Proof of Proposition 2. Fix a primitive symbol F of Σ_0. Consider a primitive application of F whose federated inputs are $X_1, \dots, X_p$ and whose shared inputs are $S_1, \dots, S_q$, and let Y denote its output.

The convention for global availability in Subsection 2.1 gives the first claim, including when the federation has one client.

We now prove the second claim. Assume that $p \geq 1$ and that Y is shared.

Assume first that $F = u$ for some $u \in \mathcal{U}$. By Definition 22, a unary primitive applied to a federated tensor produces a federated tensor. Since $p \geq 1$, the unique input is federated. Hence Y cannot be shared. This case is impossible.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$. By Definition 23, if at least one input is federated, then the output is federated. Since $p \geq 1$, at least one input is federated. Hence Y cannot be shared. This case is impossible.

Assume next that $F = \tau$ for some permutation $\tau \in \mathfrak{S}_k$. By Definition 25, a permutation primitive preserves tensor kind. Since $p \geq 1$, the unique input is federated. Hence Y cannot be shared. This case is impossible.

Assume next that $F = \alpha_j$ for some $\alpha \in \mathcal{A}$ and some axis j. Since aggregation primitives are unary, the unique input is federated because $p \geq 1$. By Definition 24, aggregation along the record axis produces a shared tensor, while aggregation along any non-record axis produces a federated tensor. Therefore, if Y is shared, there exist $\alpha \in \mathcal{A}$, an index $i \in [p]$, and the record axis r of X_i such that $Y = \alpha_r(X_i)$.

Assume next that $F = \mathsf{MatMul}_{\mathsf{FedSh}}$. By Definition 26, the output is federated. Hence Y cannot be shared. This case is impossible.

Assume next that $F = \mathsf{MatMul}_{\mathsf{ShFed}}$. By Definition 27, the output is federated. Hence Y cannot be shared. This case is impossible.

Assume finally that $F = \mathsf{MatMul}_{\mathsf{FedFed}}$. By Definition 28, the output is shared, and both operands are federated matrices. Therefore there exist indices $i, j \in [p]$ such that $Y = \mathsf{MatMul}_{\mathsf{FedFed}}(X_i, X_j)$. The two operands have record axes 2 and 1, respectively, and each federated input has one designated record axis, so $i \neq j$.

These cases exhaust the primitive symbols of Σ_0 by Definition 15. Since the aggregation symbols and the matrix-product symbols are distinct primitive symbols, the two stated alternatives are mutually exclusive. Therefore, under the assumptions $p \geq 1$ and Y shared, exactly one of the two cases holds. This proves the proposition. □

Proof of Proposition 3. Fix a primitive symbol F of Σ_0 with arity m. Consider a typed primitive application of F to inputs $A_1, \ldots, A_m$, where each A_t is either a federated tensor or a shared tensor, and let Y denote its output. For each $t \in [m]$, define $\widetilde{A}_t$ as in the statement of the proposition.

For each federated tensor X, write $\mathcal{C} = \{c_1, \ldots, c_M\}$ with $c_1 < \ldots < c_M$. If X has record axis r and local record counts $n_{c_1}(X), \ldots, n_{c_M}(X)$, define the client offset $o_X(c_j) := \sum_{\ell=1}^{j-1} n_{c_\ell}(X)$ for $j \in [M]$. If $i = (i_1, \ldots, i_k)$ is a local index of $X^{(c_j)}$, define the corresponding global index $\mu_X(c_j, i) = (h_1, \ldots, h_k)$ by $h_t = i_t$ for $t \neq r$ and $h_r = o_X(c_j) + i_r$. By Definition 4, $X^{(c_j)}[i] = \overline{X}[\mu_X(c_j, i)]$ for every local index i.

We first treat the case in which Y is federated.

Assume first that $F = u$ for some $u \in \mathcal{U}$. Then there is one federated input X_1 and no shared input. By Definition 22, $Y^{(c)} = u(X_1^{(c)})$ for every $c \in \mathcal{C}$. Let $c \in \mathcal{C}$ and let i be a local index of $Y^{(c)}$. Then $\overline{Y}[\mu_Y(c, i)] = Y^{(c)}[i] = u(X_1^{(c)}[i]) = u(\overline{X}_1[\mu_{X_1}(c, i)])$. Hence $\overline{Y} = u(\overline{X}_1) = F^{\mathrm{ord}}(\widetilde{A}_1)$.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$ and that both inputs are federated. Let the inputs be X_1 and X_2. By Definition 23, $Y^{(c)} = \beta(X_1^{(c)}, X_2^{(c)})$ for every $c \in \mathcal{C}$. Because the primitive application is typed, Definition 35 implies that X_1 and X_2 have the same federated type. Hence they have the same record axis. The semantic compatibility condition in Definition 23 gives $\mathrm{shape}(X_1^{(c)}) = \mathrm{shape}(X_2^{(c)})$ for every $c \in \mathcal{C}$. In particular, their local record counts and client offsets agree. Let i be a local index. Then $\overline{Y}[\mu_Y(c, i)] = \beta(X_1^{(c)}[i], X_2^{(c)}[i]) = \beta(\overline{X}_1[\mu_{X_1}(c, i)], \overline{X}_2[\mu_{X_2}(c, i)])$. Therefore $\overline{Y} = \beta(\overline{X}_1, \overline{X}_2) = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

Assume next that $F = \beta$ for some $\beta \in \mathcal{B} \cup \mathcal{G}$, that the first input is federated, and that the second input is shared. Let the inputs be X_1 and S_1. By Definition 23, $Y^{(c)} = \beta(X_1^{(c)}, \mathrm{Br}_c(S_1; X_1))$ for every $c \in \mathcal{C}$. Because the primitive application is typed, Definition 35 together with Definition 31 implies that

the shared tensor S_1 has extent 1 along the record axis when aligned against the symbolic shape of X_1. Hence the ordinary broadcast of S_1 against $\overline{X}_1$ is defined and, for every client c and every local index i, $\mathrm{Br}^{\mathrm{shape}(\overline{X}_1)}_{\mathrm{shape}(S_1)}(S_1)[\mu_{X_1}(c,i)] = \mathrm{Br}_c(S_1; X_1)[i]$ by Definitions 18 and 21. Therefore $\overline{Y}[\mu_Y(c,i)] = Y^{(c)}[i] = \beta(X_1^{(c)}[i], \mathrm{Br}_c(S_1; X_1)[i]) = \beta(\overline{X}_1[\mu_{X_1}(c,i)], \mathrm{Br}^{\mathrm{shape}(\overline{X}_1)}_{\mathrm{shape}(S_1)}(S_1)[\mu_{X_1}(c,i)])$. Hence $\overline{Y} = \beta(\overline{X}_1, S_1) = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

The case in which $F = \beta$, the first input is shared, and the second input is federated is identical. If the inputs are S_1 and X_1, then $\overline{Y} = \beta(S_1, \overline{X}_1) = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

Assume next that $F = \alpha_j$ for some $\alpha \in \mathcal{A}$, and that the output is federated. Then the unique input is a federated tensor X_1, and Definition 36 implies that j is not its record axis r. By Definition 24, $Y^{(c)} = \alpha_j(X_1^{(c)})$ for every $c \in \mathcal{C}$. Let i be a local index of $Y^{(c)}$. Since $j \neq r$, the fiber used to compute $Y^{(c)}[i]$ lies entirely inside the local tensor $X_1^{(c)}$. The corresponding fiber of $\overline{X}_1$ is the same fiber. Therefore $\overline{Y}[\mu_Y(c,i)] = \alpha_j(X_1^{(c)})[i] = \alpha_j(\overline{X}_1)[\mu_Y(c,i)]$. Hence $\overline{Y} = \alpha_j(\overline{X}_1) = F^{\mathrm{ord}}(\widetilde{A}_1)$.

Assume next that $F = \tau$ for some permutation $\tau \in \mathfrak{S}_k$ and that the unique input is a federated tensor X_1 of ordinary order k with record axis r. By Definition 25, $Y^{(c)} = \tau(X_1^{(c)})$ for every $c \in \mathcal{C}$, and Y has record axis $\tau(r)$. The local record counts of Y are the same as those of X_1, so $o_Y(c) = o_{X_1}(c)$ for every client c. Let $i = (i_1, \ldots, i_k)$ be a local index of $Y^{(c)}$, and define $j = (i_{\tau(1)}, \ldots, i_{\tau(k)})$. By Definition 19, $Y^{(c)}[i] = X_1^{(c)}[j]$. Let $g = \mu_Y(c,i)$. Since the record axis of Y is $\tau(r)$, we have $g_{\tau(t)} = i_{\tau(t)}$ for $t \neq r$ and $g_{\tau(r)} = o_Y(c) + i_{\tau(r)} = o_{X_1}(c) + j_r$. Hence $(g_{\tau(1)}, \ldots, g_{\tau(k)}) = \mu_{X_1}(c,j)$. Using the ordinary permutation rule of Definition 19, we obtain $\tau(\overline{X}_1)[g] = \overline{X}_1[g_{\tau(1)}, \ldots, g_{\tau(k)}] = \overline{X}_1[\mu_{X_1}(c,j)] = X_1^{(c)}[j] = Y^{(c)}[i]$. Therefore $\overline{Y} = \tau(\overline{X}_1) = F^{\mathrm{ord}}(\widetilde{A}_1)$.

Assume next that $F = \mathsf{MatMul}_{\mathsf{FedSh}}$. Then the inputs are a federated matrix X_1 with record axis 1 and a shared matrix S_1. By Definition 26, $Y^{(c)} = X_1^{(c)} S_1$ for every $c \in \mathcal{C}$. Let $N := \sum_{c \in \mathcal{C}} n_c(X_1)$. Then $\overline{X}_1 \in \mathbb{R}^{N \times p}$ and $\overline{Y} \in \mathbb{R}^{N \times q}$. Concatenation along the first axis commutes with right multiplication by the same matrix. Hence $\overline{Y} = \overline{X}_1 S_1 = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

Assume finally that $F = \mathsf{MatMul}_{\mathsf{ShFed}}$. Then the inputs are a shared matrix S_1 and a federated matrix X_1 with record axis 2. By Definition 27, $Y^{(c)} = S_1 X_1^{(c)}$ for every $c \in \mathcal{C}$. Concatenation along the second axis commutes with left multiplication by the same matrix. Therefore $\overline{Y} = S_1 \overline{X}_1 = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

This completes the case in which Y is federated.

We now treat the case in which Y is shared.

If no input is federated, then every input is shared. In that case the only possible primitive applications in the base language are shared-only applications of symbols from $\mathcal{U}$, $\mathcal{B}$, $\mathcal{G}$, $\{\alpha_j : \alpha \in \mathcal{A},\ j \geq 1\}$, and $\mathcal{P}$. By Definitions 22, 23, 24, and 25, their shared semantics agrees with the corresponding ordinary operation of Definition 19. Therefore $Y = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m)$.

Assume next that at least one input is federated. Then Proposition 2 shows that exactly one of two cases occurs.

In the first case, $Y = \alpha_r(X_1)$ for some federated tensor X_1 with record axis r. By Definition 24, $Y = \alpha_r(\overline{X}_1) = F^{\mathrm{ord}}(\widetilde{A}_1)$.

In the second case, $Y = \mathsf{MatMul}_{\mathsf{FedFed}}(X_1, X_2)$ for federated matrices X_1 and X_2 with local shapes $X_1^{(c)} \in \mathbb{R}^{a \times n_c}$ and $X_2^{(c)} \in \mathbb{R}^{n_c \times b}$ for every $c \in \mathcal{C}$. Let $N := \sum_{c \in \mathcal{C}} n_c$. Then $\overline{X}_1 \in \mathbb{R}^{a \times N}$ and $\overline{X}_2 \in \mathbb{R}^{N \times b}$. By Definition 28, $Y = \sum_{c \in \mathcal{C}} X_1^{(c)} X_2^{(c)}$. The matrix $\overline{X}_1$ is the horizontal concatenation of the blocks $X_1^{(c)}$, and the matrix $\overline{X}_2$ is the vertical concatenation of the blocks $X_2^{(c)}$ in the same client order. Therefore block multiplication gives $\overline{X}_1 \overline{X}_2 = \sum_{c \in \mathcal{C}} X_1^{(c)} X_2^{(c)}$. Hence $Y = \overline{X}_1 \overline{X}_2 = F^{\mathrm{ord}}(\widetilde{A}_1, \widetilde{A}_2)$.

These cases exhaust the primitive applications of symbols in Σ_0. Therefore, if Y is a federated tensor, then $\overline{Y} = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m)$, and if Y is a shared tensor, then $Y = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m)$. This proves the proposition. □

D Proofs for shared-state factorizations

Merge structures. Repeated application of the concatenation law in Section 3.2 gives the client-partition merge identity, including empty client blocks. The identity notation 0_i denotes the monoid identity: for product aggregation it is the all-ones tensor. In contrast, maximum on unrestricted real tensors has no empty-input identity; using a bottom element requires a restriction or enlargement of the state domain. Raw concatenation changes shape and depends on order. An expression that concatenates inputs and then reduces the result belongs to the factorization class only through a valid statistic of count-independent shape. These restrictions concern the shared-output boundaries in the program class, rather than shared-only or non-r aggregations elsewhere in the language.

Proof of Theorem 1. Let σ be induced by a one-round typed program of input type $\mathsf{Fed}_r(\mathbf{d})$ and output shape $\mathbf{t}$. Let $g_1(x), \ldots, g_q(x)$ be its shared-state expressions, and let

$$y_1 : \mathsf{Sh}(\mathbf{s}_1), \ldots, y_q : \mathsf{Sh}(\mathbf{s}_q) \vdash h(y_1, \ldots, y_q) : \mathsf{Sh}(\mathbf{t})$$

be its shared-only decoder expression.

We first record a consequence of client-locality for expressions. Let $e(x)$ be a client-local federated expression with one free variable $x : \mathsf{Fed}_r(\mathbf{d})$. We claim that, for each $n \in \mathbb{N}_0$, there is a deterministic local map $\widehat{e}_n$ defined on $\mathcal{X}_n(r, \mathbf{d})$ such that, for every finite ordered federation $\mathcal{C}$, every federated tensor $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$, and every client $c \in \mathcal{C}$, one has $e(X)^{(c)} = \widehat{e}_{n_c}(X^{(c)})$. This claim follows by induction on the syntax of e. If $e(x) = x$, take $\widehat{e}_n$ to be the identity map on $\mathcal{X}_n(r, \mathbf{d})$. For the induction step, suppose that e is obtained by applying a primitive symbol F to subexpressions that have already been given local realizations. Since e is client-local and has federated type, every primitive subexpression of e that receives a federated input has federated output. Proposition 1 then gives a deterministic client-local map for the final primitive application. Composing this map with the local realizations of the subexpressions gives the required map $\widehat{e}_n$. This proves the claim.

We now construct the shared-state factorization. For each component $i \in [q]$, define a tensor space $M_i := \mathbb{R}^{\mathbf{s}_i}$ and a commutative monoid operation $\oplus_i$ on M_i as follows.

If g_i has the record-axis aggregation form, then by Definition 6 there are a client-local federated expression $e_i(x)$, a type $\mathsf{Fed}_{r_i}(\mathbf{u}_i)$, and an aggregation schema $\alpha^{(i)} \in \mathcal{A}$ such that $g_i(x) = \alpha^{(i)}_{r_i}(e_i(x))$. The same definition supplies a commutative monoid $(\mathbb{R}^{\mathbf{s}_i}, \oplus_i, 0_i)$ satisfying the required client-wise merge identity for $\alpha^{(i)}_{r_i}$.

If g_i has the federated-federated matrix-product form, then $g_i(x) = \mathsf{MatMul}_{\mathsf{FedFed}}(a_i(x), b_i(x))$ for client-local federated expressions $a_i(x)$ and $b_i(x)$. In this case $M_i = \mathbb{R}^{m_i \times n_i}$, and we take $\oplus_i$ to be ordinary addition and 0_i to be the zero matrix.

Define $M := M_1 \times \ldots \times M_q$. Define $\oplus_M : M \times M \to M$ componentwise by

$$(z_1, \ldots, z_q) \oplus_M (z'_1, \ldots, z'_q) := (z_1 \oplus_1 z'_1, \ldots, z_q \oplus_q z'_q),$$

and define $0_M := (0_1, \ldots, 0_q)$. Since each $(M_i, \oplus_i, 0_i)$ is a commutative monoid, $(M, \oplus_M, 0_M)$ is a commutative monoid. Also, M is a finite product of tensor spaces.

For each $n \in \mathbb{N}_0$, define the local encoder $\phi_n : \mathcal{X}_n(r, \mathbf{d}) \to M$ componentwise. Let $Z \in \mathcal{X}_n(r, \mathbf{d})$.

If g_i has the record-axis aggregation form, let $\widehat{e}_{i,n}$ be the local realization of e_i constructed above. Define $\phi_{i,n}(Z) := \alpha^{(i)}_{r_i}(\widehat{e}_{i,n}(Z))$.

If g_i has the federated-federated matrix-product form, let $\widehat{a}_{i,n}$ and $\widehat{b}_{i,n}$ be the local realizations of a_i and b_i. Define $\phi_{i,n}(Z) := \widehat{a}_{i,n}(Z)\widehat{b}_{i,n}(Z)$.

Set $\phi_n(Z) := (\phi_{1,n}(Z), \ldots, \phi_{q,n}(Z))$.

Define the decoder $\psi : M \to \mathbb{R}^{\mathbf{t}}$ by evaluating the shared-only expression h on its arguments. Thus, $\psi(z_1, \ldots, z_q) := h(z_1, \ldots, z_q)$.

It remains to prove the factorization identity. Let $\mathcal{C}$ be a finite ordered federation, and let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ be a federated tensor of type $\mathsf{Fed}_r(\mathbf{d})$ with local record counts $\{n_c\}_{c\in\mathcal{C}}$. We show that, for each $i \in [q]$,

$$g_i(X) = \bigoplus_{c\in\mathcal{C}} \phi_{i,n_c}(X^{(c)}).$$

If g_i has the record-axis aggregation form, then $g_i(X) = \alpha_{r_i}^{(i)}(e_i(X))$. By the local-realization claim, $e_i(X)^{(c)} = \widehat{e}_{i,n_c}(X^{(c)})$ for every $c \in \mathcal{C}$. By the merge identity in Definition 6,

$$g_i(X) = \bigoplus_{c\in\mathcal{C}} \alpha_{r_i}^{(i)}(e_i(X)^{(c)}) = \bigoplus_{c\in\mathcal{C}} \alpha_{r_i}^{(i)}(\widehat{e}_{i,n_c}(X^{(c)})) = \bigoplus_{c\in\mathcal{C}} \phi_{i,n_c}(X^{(c)}).$$

If g_i has the federated-federated matrix-product form, then by Definition 28,

$$g_i(X) = \sum_{c\in\mathcal{C}} a_i(X)^{(c)} b_i(X)^{(c)}.$$

By the local-realization claim applied to a_i and b_i, one has $a_i(X)^{(c)} = \widehat{a}_{i,n_c}(X^{(c)})$ and $b_i(X)^{(c)} = \widehat{b}_{i,n_c}(X^{(c)})$ for every $c \in \mathcal{C}$. Hence

$$g_i(X) = \sum_{c\in\mathcal{C}} \widehat{a}_{i,n_c}(X^{(c)})\widehat{b}_{i,n_c}(X^{(c)}) = \bigoplus_{c\in\mathcal{C}} \phi_{i,n_c}(X^{(c)}),$$

where the last merge is ordinary addition in M_i.

Therefore

$$(g_1(X), \ldots, g_q(X)) = \bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})$$

in M. Applying the decoder gives

$$\sigma_{\mathcal{C}}(X) = h(g_1(X), \ldots, g_q(X)) = \psi\left(\bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})\right).$$

This is the identity required in Definition 5. Hence σ admits a shared-state factorization. □

Proof of Proposition 4. Represent the derivation $\mathcal{D}$ by a directed acyclic graph whose nodes carry typing and primitive metadata. Its size $|\mathcal{D}|$ counts nodes, edges and annotations in this representation. Merge structures are given semantic data. Shape and structure data are retained by symbolic references, with constant-time primitive metadata lookup.

Traverse the graph in dependency order, memoizing symbolic local realizations in the encoder subgraphs and retaining the shared-only decoder graph. For the input variable, emit the identity map. For a client-local primitive application, emit its local evaluation rule with references to the already compiled operands. Proposition 1 justifies this composition. The result is one family $\{\widehat{e}_n\}_{n\in\mathbb{N}_0}$ generated by the same expression and primitive rules, with n supplied by the local input shape. It contains no client-identity parameter or access to another client's data; its evaluation cost may grow with n.

For each declared boundary g_i, emit one component encoder. In the aggregation case emit $\phi_{i,n} = \alpha_{r_i}^{(i)} \circ \widehat{e}_{i,n}$ and retain its valid merge and identity. In the contraction case emit

$$\phi_{i,n}(Z) = \widehat{a}_{i,n}(Z)\widehat{b}_{i,n}(Z)$$

with addition and the zero matrix. The program's count compatibility condition ensures that this product is defined. Form the symbolic product state $M = \prod_{i=1}^{q} \mathbb{R}^{\mathbf{s}_i}$, the componentwise merge, the tuple encoder $\phi_n = (\phi_{1,n}, \ldots, \phi_{q,n})$, and the decoder $\psi = h$. The proof of Theorem 1 verifies the factorization identity.

Each node and edge is visited once and emits a bounded number of symbolic instructions or references. The output retains graph sharing, including in the decoder, so its time and space costs are $O(|\mathcal{D}|)$. Local-node memoization does not remove declared boundaries: repeated components of the tuple remain present. □

The $O(|\mathcal{D}|)$ bound concerns symbolic extraction from the provided valid derivation, without numeric tensor evaluation, type-checking, proof of merge laws, expansion of shared subexpressions into trees, enumeration of input counts, or dimension-sized array allocation. Arithmetic on arbitrarily large shape integers is also outside the constant-time metadata convention.

Proof of Corollary 1. A tensor of shape $\mathbf{s} = (s_1, \ldots, s_k)$ has $\nu(\mathbf{s}) = \prod_{j=1}^{k} s_j$ real coordinates, with one coordinate when $k = 0$. Flattening each component identifies $M = \prod_{i=1}^{q} \mathbb{R}^{\mathbf{s}_i}$ with a real vector space of dimension $\sum_{i=1}^{q} \nu(\mathbf{s}_i)$. Components are counted according to the declared tuple. Applying the same argument in round t gives $D_t(P) = \sum_{i=1}^{q_t} \nu(\mathbf{s}_{t,i})$. Appendix M gives the associated evaluation-cost accounting. □

Definition 41 (Typed-realizable shared-state factorization)**.** *A shared-state factorization of a computation σ with input type $\mathsf{Fed}_r(\mathbf{d})$ and output shape $\mathbf{t}$ is* typed-realizable *if it has state space $M = \prod_{i=1}^{q} \mathbb{R}^{\mathbf{s}_i}$, componentwise commutative monoid merges $(\mathbb{R}^{\mathbf{s}_i}, \oplus_i, 0_i)$, and the following witnesses over Σ_0.*

Each component encoder has one of two forms. For an aggregation-realized component, there are a client-local expression $e_i(x)$ and an aggregation schema $\alpha^{(i)} \in \mathcal{A}$ such that

$$x : \mathsf{Fed}_r(\mathbf{d}) \vdash e_i(x) : \mathsf{Fed}_{r_i}(\mathbf{u}_i),\ \phi_{i,n}(Z) = \alpha_{r_i}^{(i)}(\widehat{e}_{i,n}(Z))$$

for all $n \in \mathbb{N}_0$ and $Z \in \mathcal{X}_n(r, \mathbf{d})$. The output shape is $\mathbf{s}_i$, and $\alpha_{r_i}^{(i)}$ has the merge structure $(\mathbb{R}^{\mathbf{s}_i}, \oplus_i, 0_i)$ of Section 3.2.

For a matrix-product-realized component, there are client-local expressions $a_i(x), b_i(x)$ and positive integers m_i, n_i with

$$x : \mathsf{Fed}_r(\mathbf{d}) \vdash a_i(x) : \mathsf{Fed}_2((m_i)),\ x : \mathsf{Fed}_r(\mathbf{d}) \vdash b_i(x) : \mathsf{Fed}_1((n_i)).$$

The application $\mathsf{MatMul}_{\mathsf{FedFed}}(a_i(x), b_i(x))$ satisfies the client-wise count compatibility condition of Definition 28 for every input. In this case $\mathbf{s}_i = (m_i, n_i)$, the merge is addition with zero matrix identity, and

$$\phi_{i,n}(Z) = \widehat{a}_{i,n}(Z)\widehat{b}_{i,n}(Z),$$

where $n \in \mathbb{N}_0,\ Z \in \mathcal{X}_n(r, \mathbf{d})$. The maps $\widehat{e}_{i,n}, \widehat{a}_{i,n}, \widehat{b}_{i,n}$ are the uniform local families generated from their respective expressions by Proposition 1 and structural induction.

Finally, there is a shared-only decoder expression

$$y_1 : \mathsf{Sh}(\mathbf{s}_1), \ldots, y_q : \mathsf{Sh}(\mathbf{s}_q) \vdash h(y_1, \ldots, y_q) : \mathsf{Sh}(\mathbf{t})$$

that is defined on all of M and satisfies $\psi(z_1, \ldots, z_q) = h(z_1, \ldots, z_q)$ for every $(z_1, \ldots, z_q) \in M$.

A partial operation, such as a linear solve, does not provide a total map on M through shared-only typing. Its use therefore requires either a total extension or a separate domain-qualified formulation of the factorization.

Proof of Theorem 2. Let σ have a typed-realizable shared-state factorization according to Definition 41. We construct a one-round typed program representing σ.

For each component $i \in [q]$, define a shared-state expression $g_i(x)$ as follows. If the component is aggregation-realized, set $g_i(x) := \alpha_{r_i}^{(i)}(e_i(x))$. If the component is matrix-product-realized, set $g_i(x) := \mathsf{MatMul}_{\mathsf{FedFed}}(a_i(x), b_i(x))$. In both cases, $g_i(x)$ has shared type $\mathsf{Sh}(\mathbf{s}_i)$ by the typing rules of Appendix B. The decoder expression h, merge structure and contraction compatibility are given by Definition 41. These choices form a one-round typed program (see Definition 6).

It remains to show that this program induces σ. Let $\mathcal{C}$ be a finite ordered federation, and let $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ be a federated tensor of type $\mathsf{Fed}_r(\mathbf{d})$ with local record counts $\{n_c\}_{c \in \mathcal{C}}$.

Fix $i \in [q]$. If the component is aggregation-realized, then $g_i(X) = \alpha_{r_i}^{(i)}(e_i(X))$. By the assumed merge identity for $\alpha_{r_i}^{(i)}$ and by the local realization of e_i, one has

$$g_i(X) = \bigoplus_{c \in \mathcal{C}} \alpha_{r_i}^{(i)}(e_i(X)^{(c)}) = \bigoplus_{c \in \mathcal{C}} \alpha_{r_i}^{(i)}(\widehat{e}_{i,n_c}(X^{(c)})) = \bigoplus_{c \in \mathcal{C}} \phi_{i,n_c}(X^{(c)}).$$

If the component is matrix-product-realized, then $g_i(X) = \mathsf{MatMul}_{\mathsf{FedFed}}(a_i(X), b_i(X))$. By Definition 28 and by the local realizations of a_i and b_i, one has

$$g_i(X) = \sum_{c\in\mathcal{C}} a_i(X)^{(c)} b_i(X)^{(c)} = \sum_{c\in\mathcal{C}} \widehat{a}_{i,n_c}(X^{(c)})\widehat{b}_{i,n_c}(X^{(c)}) = \bigoplus_{c\in\mathcal{C}} \phi_{i,n_c}(X^{(c)}),$$

where the last merge is ordinary addition.

Thus, for every $i \in [q]$, $g_i(X) = \bigoplus_{c\in\mathcal{C}} \phi_{i,n_c}(X^{(c)})$. Since the merge on M is componentwise, this gives

$$(g_1(X), \dots, g_q(X)) = \bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)}).$$

Using the decoder identity $\psi(z_1, \dots, z_q) = h(z_1, \dots, z_q)$, we obtain

$$h(g_1(X), \dots, g_q(X)) = \psi\left(\bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})\right).$$

By the shared-state factorization identity of Definition 5, the right-hand side is $\sigma_{\mathcal{C}}(X)$. Therefore the constructed one-round typed program induces σ. This proves the theorem. □

Proof of Theorem 3. Fix a round $t \in \{0, \dots, T-1\}$. Let $g_{t,1}(x,\theta), \dots, g_{t,q_t}(x,\theta)$ be the shared-state expressions of round t, and let

$$y_1 : \mathsf{Sh}(\mathbf{s}_{t,1}), \dots, y_{q_t} : \mathsf{Sh}(\mathbf{s}_{t,q_t}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash h_t(y_1, \dots, y_{q_t}, \theta) : \mathsf{Sh}(\boldsymbol{\tau}_{t+1})$$

be the round-t shared-only decoder expression.

We first record a client-local realization claim for round-t expressions. Let $e(x,\theta)$ be a client-local federated expression with typing judgment

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash e(x,\theta) : \mathsf{Fed}_{r'}(\mathbf{u}).$$

For each fixed $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$ and each $n \in \mathbb{N}_0$, there is a deterministic local map $\widehat{e}_n(\cdot;\theta)$ on $\mathcal{X}_n(r,\mathbf{d})$ such that, for every finite ordered federation $\mathcal{C}$, every federated tensor $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$, and every client $c \in \mathcal{C}$, one has $e(X,\theta)^{(c)} = \widehat{e}_{n_c}(X^{(c)};\theta)$. The proof is by induction on the syntax of e. If $e(x,\theta) = x$, the claim holds with the identity map. In the induction step, e is obtained by applying a primitive symbol to subexpressions that already have local realizations. Since e is client-local, every primitive subexpression that receives a federated input has federated output. Proposition 1 gives a deterministic client-local map for that primitive application, with θ treated as a shared input. Composing this map with the local realizations of the subexpressions gives $\widehat{e}_n(\cdot;\theta)$.

We now construct the round-t shared-state factorization. For each $i \in [q_t]$, define $M_{t,i} := \mathbb{R}^{\mathbf{s}_{t,i}}$ and define a commutative monoid operation $\oplus_{t,i}$ on $M_{t,i}$ as follows.

If $g_{t,i}$ has the aggregation form, then Definition 7 gives a client-local federated expression $e_{t,i}(x,\theta)$, a type $\mathsf{Fed}_{r_{t,i}}(\mathbf{u}_{t,i})$, and an aggregation schema $\alpha^{(t,i)} \in \mathcal{A}$ such that $g_{t,i}(x,\theta) = \alpha^{(t,i)}_{r_{t,i}}(e_{t,i}(x,\theta))$. The same definition supplies a commutative monoid $(\mathbb{R}^{\mathbf{s}_{t,i}}, \oplus_{t,i}, 0_{t,i})$ satisfying the client-wise merge identity for $\alpha^{(t,i)}_{r_{t,i}}$.

If $g_{t,i}$ has the matrix-product form, then $g_{t,i}(x,\theta) = \mathsf{MatMul}_{\mathsf{FedFed}}(a_{t,i}(x,\theta), b_{t,i}(x,\theta))$ for client-local federated expressions $a_{t,i}$ and $b_{t,i}$. In this case, $M_{t,i} = \mathbb{R}^{m_{t,i}\times n_{t,i}}$, and we take $\oplus_{t,i}$ to be ordinary addition and $0_{t,i}$ to be the zero matrix.

Define $M_t := M_{t,1} \times \cdots \times M_{t,q_t}$. Define $\oplus_t : M_t \times M_t \to M_t$ componentwise by

$$(z_1, \dots, z_{q_t}) \oplus_t (z'_1, \dots, z'_{q_t}) := (z_1 \oplus_{t,1} z'_1, \dots, z_{q_t} \oplus_{t,q_t} z'_{q_t}),$$

and define $0_t := (0_{t,1}, \dots, 0_{t,q_t})$. Since each component is a commutative monoid, $(M_t, \oplus_t, 0_t)$ is a commutative monoid.

For each $n \in \mathbb{N}_0$ and each $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$, define the local encoder $\phi_{t,n}(\cdot;\theta) : \mathcal{X}_n(r,\mathbf{d}) \to M_t$ componentwise. Let $Z \in \mathcal{X}_n(r,\mathbf{d})$.

If $g_{t,i}$ has the aggregation form, let $\widehat{e}_{t,i,n}(\cdot;\theta)$ be the local realization of $e_{t,i}(x,\theta)$ constructed above. Define

$$\phi_{t,i,n}(Z;\theta) := \alpha^{(t,i)}_{r_{t,i}}(\widehat{e}_{t,i,n}(Z;\theta)).$$

If $g_{t,i}$ has the matrix-product form, let $\widehat{a}_{t,i,n}(\cdot;\theta)$ and $\widehat{b}_{t,i,n}(\cdot;\theta)$ be the local realizations of $a_{t,i}(x,\theta)$ and $b_{t,i}(x,\theta)$. Define

$$\phi_{t,i,n}(Z;\theta) := \widehat{a}_{t,i,n}(Z;\theta)\widehat{b}_{t,i,n}(Z;\theta).$$

Set

$$\phi_{t,n}(Z;\theta) := (\phi_{t,1,n}(Z;\theta), \dots, \phi_{t,q_t,n}(Z;\theta)).$$

Define the decoder $\psi_t : M_t \times \mathbb{R}^{\boldsymbol{\tau}_t} \to \mathbb{R}^{\boldsymbol{\tau}_{t+1}}$ by

$$\psi_t((z_1,\dots,z_{q_t}),\theta) := h_t(z_1,\dots,z_{q_t},\theta).$$

It remains to prove the factorization identity. Let $\mathcal{C}$ be a finite ordered federation, let $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ be a federated tensor of type $\mathsf{Fed}_r(\mathbf{d})$ with local record counts $\{n_c\}_{c\in\mathcal{C}}$, and let $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$.

We show that, for each $i \in [q_t]$,

$$g_{t,i}(X,\theta) = \bigoplus_{c\in\mathcal{C}} \phi_{t,i,n_c}(X^{(c)};\theta).$$

If $g_{t,i}$ has the aggregation form, then $g_{t,i}(X,\theta) = \alpha^{(t,i)}_{r_{t,i}}(e_{t,i}(X,\theta))$. By the local-realization claim, $e_{t,i}(X,\theta)^{(c)} = \widehat{e}_{t,i,n_c}(X^{(c)};\theta)$ for every $c \in \mathcal{C}$. By the merge identity in Definition 7,

$$g_{t,i}(X,\theta) = \bigoplus_{c\in\mathcal{C}} \alpha^{(t,i)}_{r_{t,i}}(e_{t,i}(X,\theta)^{(c)}) = \bigoplus_{c\in\mathcal{C}} \alpha^{(t,i)}_{r_{t,i}}(\widehat{e}_{t,i,n_c}(X^{(c)};\theta)) = \bigoplus_{c\in\mathcal{C}} \phi_{t,i,n_c}(X^{(c)};\theta).$$

If $g_{t,i}$ has the matrix-product form, then by Definition 28,

$$g_{t,i}(X,\theta) = \sum_{c\in\mathcal{C}} a_{t,i}(X,\theta)^{(c)} b_{t,i}(X,\theta)^{(c)}.$$

By the local-realization claim applied to $a_{t,i}$ and $b_{t,i}$, one has $a_{t,i}(X,\theta)^{(c)} = \widehat{a}_{t,i,n_c}(X^{(c)};\theta)$ and $b_{t,i}(X,\theta)^{(c)} = \widehat{b}_{t,i,n_c}(X^{(c)};\theta)$ for every $c \in \mathcal{C}$. Hence

$$g_{t,i}(X,\theta) = \sum_{c\in\mathcal{C}} \widehat{a}_{t,i,n_c}(X^{(c)};\theta)\widehat{b}_{t,i,n_c}(X^{(c)};\theta) = \bigoplus_{c\in\mathcal{C}} \phi_{t,i,n_c}(X^{(c)};\theta),$$

where the last merge is ordinary addition in $M_{t,i}$.

Therefore

$$(g_{t,1}(X,\theta),\dots,g_{t,q_t}(X,\theta)) = \bigoplus_{c\in\mathcal{C}} \phi_{t,n_c}(X^{(c)};\theta)$$

in M_t. Applying ψ_t gives

$$U_{t,\mathcal{C}}(X,\theta) = h_t(g_{t,1}(X,\theta),\dots,g_{t,q_t}(X,\theta),\theta) = \psi_t\left(\bigoplus_{c\in\mathcal{C}} \phi_{t,n_c}(X^{(c)};\theta),\theta\right).$$

This proves the theorem. □

E Privacy integration

The shared-state factorization $\sigma_{\mathcal{C}}(X) = \psi\left(\bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})\right)$ identifies three integration points for privacy mechanisms: the local encoder ϕ, the merge operation $\bigoplus$, and the decoder ψ. The mechanisms discussed here are not part of the deterministic base signature Σ_0. They define randomized or cryptographic realizations of the same factorization interface.

The following theorem formalizes the compatibility of shared-state factorizations with differential privacy. It is a lifting result; once the deterministic computation is expressed through the shared-state query, privacy mechanisms can be applied either to the local messages, to the merged shared state, or to the decoded output.

Theorem 5 (Privacy lifting for shared-state factorizations)**.** *Let σ admit a shared-state factorization with state space M, merge operation $\oplus_M$, local encoders $\{\phi_n\}_{n\in\mathbb{N}_0}$, and decoder ψ. For a finite ordered federation $\mathcal{C}$, define the deterministic shared-state query $Q_{\mathcal{C}}(X) := \bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})$.*

Equip the tensor spaces and their finite products with their Borel sigma-algebras. Assume that the encoders, merge, and decoder are measurable on their domains. Any restricted decoder domain is a Borel subset of M, and every randomized merged state in this appendix lies in that domain with probability one. Randomized mechanisms in this appendix are measurable probability kernels.

Let $\sim$ be an adjacency relation on federated inputs of the same type. Equip M with a metric d_M, and suppose that

$$\Delta_Q := \sup_{X\sim X'} d_M(Q_{\mathcal{C}}(X), Q_{\mathcal{C}}(X')) < \infty.$$

Let $\mathcal{R}_M : M \rightsquigarrow M$ be a randomized mechanism such that, for all $m, m' \in M$ with $d_M(m, m') \le \Delta_Q$ and all measurable $E \subseteq M$,

$$\Pr[\mathcal{R}_M(m) \in E] \le e^{\varepsilon} \Pr[\mathcal{R}_M(m') \in E] + \delta.$$

Then the randomized release $\mathcal{M}_{\mathcal{C}}(X) := \psi(\mathcal{R}_M(Q_{\mathcal{C}}(X)))$ is (ε, δ)-differentially private with respect to $\sim$.

Moreover, let $\sim_n$ be a local adjacency relation on $\mathcal{X}_n(r, \mathbf{d})$ for each $n \in \mathbb{N}_0$. Suppose that, for each n, $\mathcal{R}_n : \mathcal{X}_n(r, \mathbf{d}) \rightsquigarrow M$ is an ε-local-DP randomizer with respect to $\sim_n$. Define

$$\mathcal{M}^{\mathrm{loc}}_{\mathcal{C}}(X) := \psi\left(\bigoplus_{c\in\mathcal{C}} \mathcal{R}_{n_c}(X^{(c)})\right),$$

where the randomizers are applied independently across clients. Then $\mathcal{M}^{\mathrm{loc}}_{\mathcal{C}}$ is ε-private for replacing one client's contribution by a locally adjacent contribution under $\sim_{n_c}$, with the same client set, local sample counts, and all other client tensors unchanged.

Proof. We first prove the central-DP statement. Let $X \sim X'$ be adjacent federated inputs. By definition of Δ_Q, one has $d_M(Q_{\mathcal{C}}(X), Q_{\mathcal{C}}(X')) \le \Delta_Q$. Therefore, for every measurable $E \subseteq M$,

$$\Pr[\mathcal{R}_M(Q_{\mathcal{C}}(X)) \in E] \le e^{\varepsilon} \Pr[\mathcal{R}_M(Q_{\mathcal{C}}(X')) \in E] + \delta.$$

The released value $\mathcal{M}_{\mathcal{C}}(X) = \psi(\mathcal{R}_M(Q_{\mathcal{C}}(X)))$ is a post-processing of $\mathcal{R}_M(Q_{\mathcal{C}}(X))$. For a restricted domain, extend ψ by a constant outside it; this is measurable and leaves the release law unchanged. Differential privacy is closed under post-processing. Hence $\mathcal{M}_{\mathcal{C}}$ is (ε, δ)-differentially private with respect to $\sim$.

We now prove the local-DP statement. Fix a client $c_0 \in \mathcal{C}$. Let X and X' be federated inputs that agree on every client $c \ne c_0$, and suppose that $X^{(c_0)} \sim_{n_{c_0}} X'^{(c_0)}$. For clients $c \ne c_0$, the random messages $\mathcal{R}_{n_c}(X^{(c)})$ and $\mathcal{R}_{n_c}(X'^{(c)})$ have the same distribution. For client c_0, the ε-local-DP assumption gives the required likelihood-ratio bound for the randomized message. The merge operation and the decoder are post-processing of the collection of randomized client messages. Therefore the final released value $\mathcal{M}^{\mathrm{loc}}_{\mathcal{C}}(X)$ satisfies the same ε privacy bound for a change in the local contribution of client c_0. This proves the theorem. □

Let $\Pi_{\oplus}$ be an SMPC protocol that correctly realizes the finite merge operation induced by the commutative monoid $(M, \oplus_M, 0_M)$ [Yao, 1982, Goldreich et al., 2019, Evans et al., 2018, Keller, 2020]. Replacing the merge $\bigoplus_{c\in\mathcal{C}}$ with $\Pi_{\oplus}$ leaves the encoder ϕ and decoder ψ unchanged, and computes the same merged state $m^\star = \bigoplus_{c\in\mathcal{C}} \phi_{n_c}(X^{(c)})$. Under the protocol's threat model, its view reveals no additional information about individual summaries beyond what is implied by the ideal functionality's released values and the adversary's allowed inputs. If the functionality releases $m^\star$, this guarantee allows information implied by that state, even when $\psi(m^\star)$ reveals less.

Theorem 5 also clarifies how local differential privacy can be integrated. If $\mathcal{R}_{\mathrm{LDP}} : M \rightsquigarrow M$ is an ε-local randomizer on the state space M [Kasiviswanathan et al., 2011], one may define randomized local messages by applying $\mathcal{R}_{\mathrm{LDP}}$ to the encoded client summaries. The same merge operation $\oplus_M$ can be applied because the randomized messages still lie in M. The commutative monoid structure is not changed by this randomization, but an arbitrary local randomizer does not need to preserve the

original deterministic factorization identity. It instead defines a randomized private analogue of the original shared-state computation.

Central differential privacy can be integrated either at the shared-state level or at the final-output level [Dwork et al., 2006, Dwork and Roth, 2014]. At the shared-state level, one applies a mechanism $\mathcal{R}^{M}_{\mathrm{CDP}}$ to the merged state $m^\star$ and releases or decodes $\psi(\mathcal{R}^{M}_{\mathrm{CDP}}(m^\star))$. At the final-output level, one applies a mechanism $\mathcal{R}^{\mathrm{out}}_{\mathrm{CDP}}$ to $\psi(m^\star)$. In both cases, the privacy guarantee applies to the quantity that is released. If the unnoised shared state $m^\star$ is revealed, then subsequent randomized post-processing cannot make that earlier release private.

These results show formal compatibility between shared-state factorizations and standard privacy mechanisms. An analysis of privacy parameters, sensitivities, cryptographic assumptions, and protocol overhead is beyond the scope of this paper. The local result concerns replacement adjacency at unchanged sample counts. It supplies no privacy bound for sample addition or removal, repeated rounds, or randomized participation without further assumptions and analysis.

F Closure under shared-only post-processing

Corollary 4 (Closure under shared-only post-processing)**.** *Let $\sigma_1, \dots, \sigma_\ell$ be shared-output computations with common input type $\mathsf{Fed}_r(\mathbf{d})$. Assume that each σ_j is induced by a one-round typed program and has output shape $\mathbf{t}_j$. Let $z_1 : \mathsf{Sh}(\mathbf{t}_1), \dots, z_\ell : \mathsf{Sh}(\mathbf{t}_\ell) \vdash H(z_1, \dots, z_\ell) : \mathsf{Sh}(\mathbf{t})$ be a shared-only expression. Define $\sigma_{\mathcal{C}}(X) := H(\sigma_{1,\mathcal{C}}(X), \dots, \sigma_{\ell,\mathcal{C}}(X))$ for every finite ordered federation $\mathcal{C}$ and every federated input X of type $\mathsf{Fed}_r(\mathbf{d})$. Then σ is induced by a one-round typed program. In particular, σ admits a shared-state factorization.*

Proof of Corollary 4. For each $j \in [\ell]$, let the one-round typed program inducing σ_j have shared-state expressions $g_{j,1}(x), \dots, g_{j,q_j}(x)$ with output shapes $\mathbf{s}_{j,1}, \dots, \mathbf{s}_{j,q_j}$, and let its shared-only decoder be

$$y_{j,1} : \mathsf{Sh}(\mathbf{s}_{j,1}), \dots, y_{j,q_j} : \mathsf{Sh}(\mathbf{s}_{j,q_j}) \vdash h_j(y_{j,1}, \dots, y_{j,q_j}) : \mathsf{Sh}(\mathbf{t}_j).$$

Construct a new one-round typed program by taking all shared-state expressions from all the programs. Thus, the shared-state expressions of the new program are $\{g_{j,a}(x) : j \in [\ell],\ a \in [q_j]\}$. Each of these expressions has one of the two forms allowed in Definition 6, because it came from one of the original one-round typed programs.

Define the new shared-only decoder by

$$\widetilde{h} := H\left(h_1(y_{1,1}, \dots, y_{1,q_1}), \dots, h_\ell(y_{\ell,1}, \dots, y_{\ell,q_\ell})\right).$$

This is a shared-only expression because each h_j is shared-only and H is shared-only.

Let $\mathcal{C}$ be a finite ordered federation and let X be a federated input of type $\mathsf{Fed}_r(\mathbf{d})$. The value induced by the constructed program is

$$\widetilde{h}\big(g_{1,1}(X), \dots, g_{\ell,q_\ell}(X)\big) = H\left(h_1(g_{1,1}(X), \dots, g_{1,q_1}(X)), \dots, h_\ell(g_{\ell,1}(X), \dots, g_{\ell,q_\ell}(X))\right).$$

Since the j-th original program induces σ_j, this equals

$$H(\sigma_{1,\mathcal{C}}(X), \dots, \sigma_{\ell,\mathcal{C}}(X)) = \sigma_{\mathcal{C}}(X).$$

Therefore σ is induced by a one-round typed program. The final claim follows from Theorem 1. □

The corollary lets the language express composite statistics without introducing new shared-state primitives. For a positive total count, a mean follows from a sum and count by shared-only division, and a variance follows from a sum, sum of squares, and count by shared-only algebra. Total decoder maps also specify their values outside that domain, as in the mean example of Section 2.

G Scope of the base iterative class

The base iterative class carries information across rounds only through shared tensors. It covers algorithms whose server-side state is updated from client-local tensor computations and shared aggregations.

Proposition 5 (Round-wise shared memory)**.** *Let an iterative typed program be given as in Definition 7. For every federated input X and every round $t \in \{0, \dots, T\}$, the iterate θ_t is a shared tensor of type $\mathsf{Sh}(\boldsymbol{\tau}_t)$.*

Moreover, let $e(x, \theta)$ be any client-local federated expression appearing in round t of the program, with typing judgment $x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash e(x, \theta) : \mathsf{Fed}_{r'}(\mathbf{u})$. For each fixed $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$ and each $n \in \mathbb{N}_0$, there exists a deterministic map $\widehat{e}_n(\cdot; \theta)$ such that, for every federated input $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ and every client $c \in \mathcal{C}$ with local record count n_c, one has $e(X, \theta)^{(c)} = \widehat{e}_{n_c}(X^{(c)}; \theta)$.

Proof of Proposition 5. We first prove that θ_t is shared for every $t \in \{0, \dots, T\}$. By Definition 7, the initial state satisfies $\theta_0 \in \mathbb{R}^{\boldsymbol{\tau}_0}$ and has type $\mathsf{Sh}(\boldsymbol{\tau}_0)$. Suppose that θ_t is shared of type $\mathsf{Sh}(\boldsymbol{\tau}_t)$. For each $i \in [q_t]$, the expression $g_{t,i}(X, \theta_t)$ is a shared-state expression and therefore has shared type $\mathsf{Sh}(\mathbf{s}_{t,i})$. The decoder has typing judgment

$$y_1 : \mathsf{Sh}(\mathbf{s}_{t,1}), \dots, y_{q_t} : \mathsf{Sh}(\mathbf{s}_{t,q_t}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash h_t(y_1, \dots, y_{q_t}, \theta) : \mathsf{Sh}(\boldsymbol{\tau}_{t+1}).$$

Thus $\theta_{t+1} = h_t(g_{t,1}(X, \theta_t), \dots, g_{t,q_t}(X, \theta_t), \theta_t)$ is shared of type $\mathsf{Sh}(\boldsymbol{\tau}_{t+1})$. The claim follows by induction on t.

We now prove the local-realization statement. Fix a round t, fix $\theta \in \mathbb{R}^{\boldsymbol{\tau}_t}$, and let $e(x, \theta)$ be a client-local federated expression with typing judgment

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}_t) \vdash e(x, \theta) : \mathsf{Fed}_{r'}(\mathbf{u}).$$

We prove by structural induction on e that, for each $n \in \mathbb{N}_0$, there exists a deterministic map $\widehat{e}_n(\cdot; \theta)$ with the stated property.

If $e(x, \theta) = x$, take $\widehat{e}_n(\cdot; \theta)$ to be the identity map on $\mathcal{X}_n(r, \mathbf{d})$. Then $e(X, \theta)^{(c)} = X^{(c)} = \widehat{e}_{n_c}(X^{(c)}; \theta)$.

For the induction step, suppose that e is obtained by applying a primitive symbol F to subexpressions $e_1, \dots, e_m$ and shared arguments formed from θ and any fixed shared constants in the program. Since e is client-local and has federated output, the final primitive application has federated output whenever it receives a federated input. By the induction hypothesis, every federated subexpression e_j has deterministic local realizations $\widehat{e}_{j,n}(\cdot; \theta)$. Proposition 1 gives a deterministic map for the local output of the final primitive application as a function of the local values of its federated inputs and its shared inputs. Composing this deterministic map with the local realizations $\widehat{e}_{j,n}(\cdot; \theta)$ gives a deterministic map $\widehat{e}_n(\cdot; \theta)$ such that $e(X, \theta)^{(c)} = \widehat{e}_{n_c}(X^{(c)}; \theta)$ for every federated input X and every client $c \in \mathcal{C}$.

This completes the induction and proves the proposition. □

Proposition 5 gives a boundary of the present language. A client may compute client-local tensor expressions allowed by the typing rules inside a round, but the only information that persists to later rounds is the shared iterate θ_t. Algorithms with persistent private client variables require an extension of the type system, for example a separate local-state sort.

H Transient intra-round state

This appendix extends the program interface. The tensor signature of Proposition 7 remains unchanged. A local encoder may retain temporary private summaries and model iterates within a round, then send a tensor message to a designated merge. These summaries and iterates are discarded before the next round. This permits local optimization without adding persistent private state to the language.

For each round t, let $M_t = \mathbb{R}^{\mathbf{s}_{t,1}} \times \ldots \times \mathbb{R}^{\mathbf{s}_{t,q_t}},\ q_t \geq 1$, where the shapes $\mathbf{s}_{t,i}$ do not depend on the numbers of clients or samples. Equip M_t with a commutative monoid $(M_t, \oplus_t, 0_t)$. The interface consists of a deterministic encoder family $E_{t,n} : \mathcal{X}_n(r, \mathbf{d}) \times \mathbb{R}^{\boldsymbol{\tau}_t} \longrightarrow M_t,\ n \in \mathbb{N}_0$, and a shared-only decoder $H_t : M_t \times \mathbb{R}^{\boldsymbol{\tau}_t} \longrightarrow \mathbb{R}^{\boldsymbol{\tau}_{t+1}}$. The family uses one local computation rule for all clients: a client with n samples applies $E_{t,n}$ to its own data and the current shared state. It has no access to another client's data, temporary state, or message. Only the result of the designated merge is made shared. The interface has no operation that releases another intermediate local value. All private state used by $E_{t,n}$ is initialized from its inputs and discarded after the round.

Let $\mathcal{C}_t \subseteq \mathcal{C}$ be a finite nonempty participating set. For the round conditioned on this set, define

$$\theta_{t+1} = H_t\left(\bigoplus_{c\in\mathcal{C}_t} E_{t,n_c}(X^{(c)};\theta_t),\theta_t\right). \tag{1}$$

Sampling $\mathcal{C}_t$, if used, is external to this deterministic update. This equation describes the update on the participating data, and it does not identify a sampled gradient with the gradient of the full federation.

Proposition 6 (Transient-encoder factorization)**.** *For the round update in Equation 1, assume that M_t has count-independent tensor shapes and a commutative monoid merge, that the encoders use one client-local computation rule with private state initialized from their inputs and discarded after each round, and that only the designated merge produces shared state. With a shared-only decoder H_t, the round factors through M_t with local encoders $\phi_{t,n} = E_{t,n}$ and decoder $\psi_t = H_t$. Each client's temporary state depends only on its own data and the current shared state, and shared-state formation occurs only through the designated merge. No private state persists across rounds.*

Proof. Substituting $\phi_{t,n} = E_{t,n}$ and $\psi_t = H_t$ into Equation 1 gives the factorization. The encoder family provides a map for every local sample count, independent of client identity. Its inputs restrict the data available to each client, and its codomain has dimension independent of the client and sample counts. The interface conditions restrict shared-state formation to the merge and discard temporary private state after the round. □

The exposure condition in this interface concerns which local values become shared. Proposition 2 retains its primitive classification for the syntax; the present interface adds a private-message merge path. Proposition 3 also does not extend to this interface in general, as the weighted FedAvg and FedProx example below shows.

Example 2 (Weighted FedAvg and FedProx)**.** *Let the shared parameter lie in $\mathbb{R}^p$, and let $K \geq 1$ be the number of local steps in a round. For $n > 0$, assume a real-valued loss $\ell(z, w)$ on the sample space and $\mathbb{R}^p$, differentiable in w, and define the local mean objective*

$$F_n(Z, w) = \frac{1}{n}\sum_{a=1}^{n} \ell(Z_a, w).$$

The notation Z_a denotes the a-th sample along axis r. Initialize $w_{t,0}(Z;\theta) = \theta$. For example, full-batch local gradient descent uses

$$w_{t,k+1} = w_{t,k} - \eta_{t,k}\nabla_w F_n(Z, w_{t,k}),\ k = 0,\dots,K-1.$$

FedProx uses the same construction with objective

$$F_n(Z, w) + \frac{\mu}{2}\|w-\theta\|^2,\ \mu \geq 0.$$

The proximal center is the current shared parameter, so this construction requires no private state from an earlier round.

For either local objective, use $M_t = \mathbb{R}^p \times \mathbb{R}$ with ordinary addition and identity $(\mathbf{0}, 0)$. Define

$$E_{t,n}(Z;\theta) = \begin{cases} (n\, w_{t,K}(Z;\theta), n), & n > 0, \\ (\mathbf{0}, 0), & n = 0, \end{cases}$$

and

$$H_t((A, N),\theta) = \begin{cases} A/N, & N > 0, \\ \theta, & N \leq 0. \end{cases}$$

An empty client sends the merge identity without evaluating a local mean objective. On messages produced by the encoders, $N = \sum_{c\in\mathcal{C}_t} n_c \geq 0$. For $N > 0$, the decoder gives the sample-weighted average of the local models. For $N = 0$, it retains the current shared parameter. The $N \leq 0$ branch extends the decoder to all of $M_t \times \mathbb{R}^p$.

All K local steps occur inside one encoder before one merge. Thus, one logical communication round comprises distribution of the shared model, local computation, and message aggregation; T counts these rounds rather than individual local steps. Physical network scheduling is separate from this semantic interface. Stochastic local steps can instead be represented by a random encoder, using the same computation rule with an additional local random seed. Conditional on those seeds, Equation 1 gives the corresponding deterministic update. This does not add a convergence or privacy result.

Example 3 (Dependence on the data partition)**.** *Consider scalar losses $\ell_h(w) = hw^2/2$ for samples $h = 1, 3$, with initial parameter $\theta = 1$, step size $\eta = 1/4$, and $K = 2$. Two clients with one sample each produce*

$$w_{1,2} = (1 - 1/4)^2 = 9/16,\ w_{3,2} = (1 - 3/4)^2 = 1/16.$$

Their sample-weighted average is $5/16$. A single client containing both samples has mean objective $F_2(w) = w^2$, so two local steps give $(1 - 2/4)^2 = 1/4$. Both partitions have the same virtual global input, but their updates differ. This prevents a general virtual global-consistency conclusion for the transient interface.

Methods that retain client-specific control variables between rounds, such as SCAFFOLD or FedDyn, require persistent local state and participation-dependent state updates. They are outside both the base iterative class and the transient interface of this appendix.

I Proofs for learning updates

Proof of Theorem 4. Let (ℓ, γ) be a representably differentiable loss. Fix a finite ordered federation $\mathcal{C}$ and a federated input $X = \{X^{(c)}\}_{c\in\mathcal{C}}$ of type $\mathsf{Fed}_r(\mathbf{d})$, with local record counts $\{n_c\}_{c\in\mathcal{C}}$.

By Definition 8, for each client $c \in \mathcal{C}$, each record index $a \in [n_c]$, and each $\theta \in \mathbb{R}^{\boldsymbol{\tau}}$, the map $\theta \mapsto \widehat{\ell}_{n_c}(X^{(c)}; \theta)[a]$ is differentiable and satisfies

$$\nabla_\theta \widehat{\ell}_{n_c}(X^{(c)}; \theta)[a] = \widehat{\gamma}_{n_c}(X^{(c)}; \theta)[a, \cdot].$$

The empirical objective $L_{\mathcal{C}}(X, \theta)$ is a finite sum of these differentiable scalar functions. Therefore it is differentiable, and the gradient of a finite sum is the sum of the gradients. Hence

$$\nabla_\theta L_{\mathcal{C}}(X, \theta) = \sum_{c\in\mathcal{C}} \sum_{a=1}^{n_c} \nabla_\theta \widehat{\ell}_{n_c}(X^{(c)}; \theta)[a] = \sum_{c\in\mathcal{C}} \sum_{a=1}^{n_c} \widehat{\gamma}_{n_c}(X^{(c)}; \theta)[a, \cdot] = G_{\mathcal{C}}(X, \theta).$$

It remains to prove the expressibility claim. By the typing assumption in Definition 8,

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash \gamma(x, \theta) : \mathsf{Fed}_1(\boldsymbol{\tau}).$$

Since $\mathrm{Sum} \in \mathcal{A}$, the aggregation typing rule gives

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash \mathrm{Sum}_1(\gamma(x, \theta)) : \mathsf{Sh}(\boldsymbol{\tau}).$$

For any X and θ, Definition 24 gives

$$\mathrm{Sum}_1(\gamma(X, \theta)) = \mathrm{Sum}_1(\overline{\gamma(X, \theta)}).$$

By the local realization of the client-local expression γ, the local tensor $\gamma(X, \theta)^{(c)}$ is equal to $\widehat{\gamma}_{n_c}(X^{(c)}; \theta)$ for every client c. Therefore

$$\mathrm{Sum}_1(\gamma(X, \theta)) = \sum_{c\in\mathcal{C}} \sum_{a=1}^{n_c} \widehat{\gamma}_{n_c}(X^{(c)}; \theta)[a, \cdot] = G_{\mathcal{C}}(X, \theta).$$

The merge operation for this record-axis summation is ordinary addition in $\mathbb{R}^{\boldsymbol{\tau}}$. Thus the gradient computation is represented by the shared-state expression $g(x, \theta) = \mathrm{Sum}_1(\gamma(x, \theta))$ and has the claimed round-wise shared-state form. □

Proof of Corollary 2. We construct an iterative typed program inducing the stated update. Fix a round $t \in \{0, \ldots, T - 1\}$. The current shared state has type $\mathsf{Sh}(\boldsymbol{\zeta}_t)$. The shared-only expression P_t maps it to a parameter of type $\mathsf{Sh}(\boldsymbol{\tau})$.

By Definition 8, the per-record gradient expression has typing judgment

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}(\boldsymbol{\tau}) \vdash \gamma(x, \theta) : \mathsf{Fed}_1(\boldsymbol{\tau}).$$

Substituting the shared-only expression $P_t(s)$ for θ gives a client-local federated expression

$$x : \mathsf{Fed}_r(\mathbf{d}), s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash \gamma(x, P_t(s)) : \mathsf{Fed}_1(\boldsymbol{\tau}).$$

Define the round-t shared-state expression

$$g_t(x, s) := \mathrm{Sum}_1(\gamma(x, P_t(s))).$$

By the aggregation typing rule, $g_t(x, s)$ has type $\mathsf{Sh}(\boldsymbol{\tau})$. It has the aggregation form allowed in Definition 7, with ordinary addition as the merge operation.

Let the round-t decoder be

$$h_t(y, s) := H_t(y, s).$$

This is a shared-only expression of type $\mathsf{Sh}(\boldsymbol{\zeta}_{t+1})$ by assumption. Therefore the expressions g_t and h_t define a valid round of an iterative typed program. Repeating this construction for $t = 0, \ldots, T-1$ gives an iterative typed program with initial state s_0.

It remains to identify the induced update. Let $\mathcal{C}$ be a finite ordered federation, let $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ be a federated input of type $\mathsf{Fed}_r(\mathbf{d})$, and let $s_t \in \mathbb{R}^{\boldsymbol{\zeta}_t}$ be the current shared optimizer state. By Theorem 4, the shared-state expression $g_t(X, s_t) = \mathrm{Sum}_1(\gamma(X, P_t(s_t)))$ evaluates to $G_{\mathcal{C}}(X, P_t(s_t))$. Hence the round update induced by the program is

$$s_{t+1} = h_t(g_t(X, s_t), s_t) = H_t(G_{\mathcal{C}}(X, P_t(s_t)), s_t).$$

This is the update in the statement. Therefore $X \mapsto s_T$ is induced by an iterative typed program. The final claim follows from Theorem 3. □

Proof of Corollary 3. We construct an iterative typed program inducing the stated update. Fix a round $t \in \{0, \ldots, T-1\}$. The current shared state has type $\mathsf{Sh}(\boldsymbol{\zeta}_t)$, and the shared-only expression P_t maps it to a parameter of type $\mathsf{Sh}((p))$.

By Definition 8, the per-record gradient expression has typing judgment

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}((p)) \vdash \gamma(x, \theta) : \mathsf{Fed}_1((p)).$$

Substituting the shared-only expression $P_t(s)$ for θ gives

$$x : \mathsf{Fed}_r(\mathbf{d}), s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash \gamma(x, P_t(s)) : \mathsf{Fed}_1((p)).$$

Define the gradient shared-state expression

$$g_t(x, s) := \mathrm{Sum}_1(\gamma(x, P_t(s))).$$

By the aggregation typing rule, $g_t(x, s)$ has type $\mathsf{Sh}((p))$ and has the aggregation form allowed in Definition 7.

Similarly, the curvature expression B has typing judgment

$$x : \mathsf{Fed}_r(\mathbf{d}), \theta : \mathsf{Sh}((p)) \vdash B(x, \theta) : \mathsf{Fed}_1((p, p)).$$

Substituting $P_t(s)$ for θ gives

$$x : \mathsf{Fed}_r(\mathbf{d}), s : \mathsf{Sh}(\boldsymbol{\zeta}_t) \vdash B(x, P_t(s)) : \mathsf{Fed}_1((p, p)).$$

Define the curvature shared-state expression

$$C_t(x, s) := \mathrm{Sum}_1(B(x, P_t(s))).$$

By the aggregation typing rule, $C_t(x, s)$ has type $\mathsf{Sh}((p, p))$ and has the aggregation form allowed in Definition 7.

Let the round-t decoder be $h_t(y, C, s) := H_t(y, C, s)$. This is a shared-only expression of type $\mathsf{Sh}(\boldsymbol{\zeta}_{t+1})$ by assumption. The shared-linear-algebra primitives used inside H_t act only on shared tensors, so they are part of the shared-only decoder and do not affect the client-local expressions. Thus g_t, C_t, and h_t define a valid round of an iterative typed program in the shared-linear-algebra extension of the language. Repeating this construction for $t = 0, \ldots, T-1$ gives an iterative typed program with initial state s_0.

It remains to identify the induced update. Let $\mathcal{C}$ be a finite ordered federation, let $X = \{X^{(c)}\}_{c \in \mathcal{C}}$ be a federated input of type $\mathsf{Fed}_r(\mathbf{d})$, and let $s_t \in \mathbb{R}^{\boldsymbol{\zeta}_t}$ be the current shared optimizer state. By

Theorem 4, the expression $g_t(X, s_t)$ evaluates to $G_{\mathcal{C}}(X, P_t(s_t))$. By Definition 24 and the local realization of B, the expression $C_t(X, s_t)$ evaluates to

$$\sum_{c\in\mathcal{C}}\sum_{a=1}^{n_c}\widehat{B}_{n_c}(X^{(c)}; P_t(s_t))[a,\cdot,\cdot] = C_{\mathcal{C}}(X, P_t(s_t)).$$

Therefore the round update induced by the program is

$$s_{t+1} = h_t(g_t(X, s_t), C_t(X, s_t), s_t) = H_t(G_{\mathcal{C}}(X, P_t(s_t)), C_{\mathcal{C}}(X, P_t(s_t)), s_t).$$

This is the update in the statement. Therefore $X \mapsto s_T$ is induced by an iterative typed program. The final claim follows from Theorem 3, whose proof applies because the added linear-algebra primitives are shared-only. □

J Learning examples

This appendix presents two learning examples for Section 4. We use client-local feature and response projections and the following operations under Proposition 7. For $U : \mathsf{Fed}_1(\varnothing)$ and $V : \mathsf{Fed}_1((p))$ with the same local sample counts and sample order, define

$$\mathrm{ScaleRec}(U, V)^{(c)}[a, j] := U^{(c)}[a]V^{(c)}[a, j].$$

Its output has type $\mathsf{Fed}_1((p))$. Its ordinary interpretation reshapes the scalar-sample vector from $(n,)$ to $(n, 1)$ and multiplies it elementwise with the feature matrix of shape (n, p). For $V : \mathsf{Fed}_1((p))$ and $\theta : \mathsf{Sh}((p))$, define

$$\mathrm{Pred}(V, \theta)^{(c)}[a] := \sum_{j=1}^{p} V^{(c)}[a, j]\theta_j,$$

of type $\mathsf{Fed}_1(\varnothing)$. Its ordinary interpretation is the row-wise dot product. It can also be represented by a federated-shared matrix product after reshaping θ to $(p, 1)$ and removing the output's non-sample singleton axis. We also use the sample-wise outer product $\mathrm{Outer}(v, v)$, defined in this appendix. Each operation processes samples independently using the same rule at every client. For aligned sample blocks, concatenating the client outputs gives the same result as applying the ordinary tensor operation to the concatenated inputs (Appendix K).

Example 4 (Logistic regression with server-side first-order optimization)**.** *Consider binary logistic regression with covariates $x_a \in \mathbb{R}^p$, responses $y_a \in \{0, 1\}$, and parameter $\theta \in \mathbb{R}^p$. Let $\rho(t) = (1 + \exp(-t))^{-1}$. The per-record negative log-likelihood and gradient are $\ell_a(\theta) = \log(1 + \exp(x_a^\top\theta)) - y_a x_a^\top\theta$ and $\gamma_a(\theta) = (\rho(x_a^\top\theta) - y_a)x_a$. After adding the scalar maps used in the loss and gradient, coordinate projections, and the aligned operations* ScaleRec *and* Pred*, these are client-local tensor expressions. Hence (ℓ, γ) is a representably differentiable loss. By Theorem 4, the global gradient is the record-axis sum of the federated gradient tensor. By Corollary 2, gradient descent, server-side momentum, and server-side Adam updates for this model are induced by iterative typed programs.*

Let the federated record tensor contain, for each record, a covariate vector in $\mathbb{R}^p$ and a binary response. We write X_{feat} and Y for the client-local projections onto covariates and responses. These projections are conservative client-local primitives in the sense of Proposition 7.

Let $\theta : \mathsf{Sh}((p))$ and define

$$R(X, \theta) := \mathrm{Pred}(X_{\mathrm{feat}}, \theta),\ P(X, \theta) := \rho(R(X, \theta)).$$

The gradient tensor is

$$\gamma(X, \theta) := \mathrm{ScaleRec}(P(X, \theta) - Y, X_{\mathrm{feat}}).$$

The residual and features are aligned projections of the same input, so the expression has type $\mathsf{Fed}_1((p))$. Thus $\mathrm{Sum}_1(\gamma(X, \theta)) : \mathsf{Sh}((p))$ evaluates to the global logistic-regression gradient. Server-side gradient descent, momentum, and Adam follow from Corollary 2 with the corresponding shared-only update expression.

Example 5 (Gaussian linear regression with a damped Newton update)**.** *Consider the Gaussian linear model $y_a \mid x_a, \theta \sim \mathcal{N}(x_a^\top\theta, \sigma^2)$ with covariates $x_a \in \mathbb{R}^p$, responses $y_a \in \mathbb{R}$, parameter $\theta \in \mathbb{R}^p$, and known variance $\sigma^2 > 0$. Up to an additive constant, the per-record negative log-likelihood*

is $\ell_a(\theta) = (y_a - x_a^\top\theta)^2/(2\sigma^2)$. The per-record gradient is $\gamma_a(\theta) = (x_a^\top\theta - y_a)x_a/\sigma^2$, and the per-record curvature block is $B_a = x_a x_a^\top/\sigma^2$. Therefore

$$G_{\mathcal{C}}(X,\theta) = \sum_{c\in\mathcal{C}}\sum_{a=1}^{n_c}\frac{1}{\sigma^2}(x_{c,a}^\top\theta - y_{c,a})x_{c,a}$$

and

$$C_{\mathcal{C}}(X,\theta) = \sum_{c\in\mathcal{C}}\sum_{a=1}^{n_c}\frac{1}{\sigma^2}x_{c,a}x_{c,a}^\top.$$

Both are record-axis sums of client-local tensor expressions, so the damped Newton step

$$\theta^+ = \theta - \eta\ \mathrm{Solve}(C_{\mathcal{C}}(X,\theta) + \lambda I, G_{\mathcal{C}}(X,\theta))$$

is an instance of Corollary 3, when the shared linear system is defined.

Let the federated record tensor contain, for each record, a covariate vector in $\mathbb{R}^p$ and a real-valued response. We again write X_{feat} and Y for the client-local projections onto covariates and responses. Consider the Gaussian linear model $y_a \mid x_a, \theta \sim \mathcal{N}(x_a^\top\theta, \sigma^2)$, with known $\sigma^2 > 0$.

Let $\theta : \mathsf{Sh}((p))$ and define the residual

$$R(X,\theta) := \mathrm{Pred}(X_{\mathrm{feat}},\theta) - Y.$$

The gradient tensor is

$$\gamma(X,\theta) := \frac{1}{\sigma^2}\,\mathrm{ScaleRec}(R(X,\theta), X_{\mathrm{feat}}),$$

which has type $\mathsf{Fed}_1((p))$. Let $\mathrm{Outer}(v,v) \in \mathbb{R}^{p\times p}$ denote the matrix with entries $\mathrm{Outer}(v,v)_{jk} = v_j v_k$. The per-record curvature tensor is

$$B(X,\theta)[a,:,:] := \frac{1}{\sigma^2}\,\mathrm{Outer}(X_{\mathrm{feat}}[a,:], X_{\mathrm{feat}}[a,:]),$$

which has type $\mathsf{Fed}_1((p,p))$. Thus $\mathrm{Sum}_1(\gamma(X,\theta))$ and $\mathrm{Sum}_1(B(X,\theta))$ are shared tensors of types $\mathsf{Sh}((p))$ and $\mathsf{Sh}((p,p))$. A damped Newton decoder is the shared-only expression

$$H(g,C,\theta) = \theta - \eta\ \mathrm{Solve}(C + \lambda I, g),$$

defined on the domain where $C + \lambda I$ is non-singular. This is the form covered by Corollary 3.

K Conservative extensions of the primitive signature

The base signature is intentionally small. It contains only the primitive families needed to state the core separation between client-local computation and shared-state formation. This keeps the semantics of Section 2 and the factorization results of Section 3 independent of model-specific operations. Applications may require coordinate projections, reshaping, row-wise maps, convolutions, or model-specific differentiable maps. Their use as conservative extensions requires the locality and ordinary-semantics conditions of Proposition 7. A convolution using samples across client boundaries fails these conditions. They may also need shared-only primitives, such as matrix multiplication or linear solves. The factorization results do not depend on the particular list of such primitives. They depend on whether the added primitives preserve the same separation between client-local computation and shared-state formation.

Proposition 7 (Conservative signature extensions)**.** *Let Σ^+ be an extension of the base signature Σ_0. Assume that every primitive symbol in $\Sigma^+ \setminus \Sigma_0$ is of one of the following two kinds.*

First, it may be shared-only. In this case it is defined only on shared inputs and returns a shared tensor.

Second, it may be client-local. In this case, whenever it receives at least one federated input, it returns a federated tensor, and for every typed primitive application with federated output Y there is, for each client $c \in \mathcal{C}$, a deterministic map Φ_c such that $Y^{(c)} = \Phi_c(X_1^{(c)}, \ldots, X_p^{(c)}, S_1, \ldots, S_q)$, where $X_1, \ldots, X_p$ are the federated inputs and $S_1, \ldots, S_q$ are the shared inputs of that primitive

application. The maps Φ_c may depend on the primitive symbol, the operand positions, and the local input shapes at client c, but not on the identity of c or on any tensor stored at another client.

Each added symbol F has an ordinary interpretation F^{ord} defined on the ordinary inputs corresponding to its permitted typed applications. An all-shared application of a client-local symbol, if permitted, returns a shared tensor. For every added primitive application $Y = F(A_1, \ldots, A_m)$, set

$$\widetilde{A}_i = \begin{cases} \overline{A_i}, & A_i \text{ is federated}, \\ A_i, & A_i \text{ is shared}. \end{cases}$$

Require ordinary agreement on the application's domain:

$$\begin{cases} \overline{Y} = F^{\mathrm{ord}}(\widetilde{A}_1, \ldots, \widetilde{A}_m), & Y \text{ is federated}, \\ Y = F^{\mathrm{ord}}(A_1, \ldots, A_m), & Y \text{ is shared}. \end{cases}$$

Any structural alignment conditions and domains of partial operations are part of the permitted application.

Then Propositions 1, 2, and 3 remain valid for typed primitive applications in Σ^+. Moreover, Theorems 1 and 3 remain valid for one-round and iterative typed programs over Σ^+, provided shared-state formation occurs only through the record-axis elimination primitives allowed in Definitions 6 and 7. For this theorem-preservation conclusion, the programs' local encoders and shared decoders are defined on the full input and state domains in those theorems and propositions, including all shared parameter values in the iterative case. Operations with restricted domains use total extensions.

Packing and component extraction from Subsection 3.3 are total shared-only primitives. Taking their ordinary interpretations to be the same packing and extraction operations satisfies the premises of Proposition 7.

Proof of Proposition 7. We first prove the three primitive semantic propositions for typed primitive applications in Σ^+.

For symbols in the base signature Σ_0, the claims are Propositions 1, 2, and 3.

Let $F \in \Sigma^+ \setminus \Sigma_0$. An all-shared application returns a shared tensor and agrees with F^{ord} by the ordinary-agreement assumption. An application with federated inputs returns a federated tensor and has the assumed uniform local realization. Its virtual global tensor equals F^{ord} applied to the ordinary inputs by the same assumption. Thus, the added symbols satisfy uniform locality and virtual-global consistency and create no new federated-to-shared boundary.

Therefore Propositions 1, 2, and 3 hold for every typed primitive application in Σ^+.

It remains to prove that the shared-state factorization theorems remain valid. The proof of Theorem 1 uses the primitive semantics only through the client-local realization of client-local federated expressions and through the specified merge identities of the shared-state expressions. For programs over Σ^+, the same client-local realization follows by structural induction on the expression. The induction step uses the extended version of Proposition 1, proved above. The rest of the proof is unchanged, because shared-state formation is assumed to occur only through the record-axis elimination primitives allowed in Definition 6.

The same argument applies to Theorem 3. Its proof uses primitive semantics only through client-local realization of round-wise federated expressions and through the prescribed merge identities of the shared-state expressions. The extended client-locality proposition gives the same local realization by structural induction, with the current shared iterate treated as a shared input. Since the allowed shared-state formation mechanisms are unchanged, the encode, merge, and decode construction is identical.

The full-domain premise makes the constructed encoder and decoder maps total on the domains required by the two theorems. Thus, both factorization theorems remain valid with the same valid shared-output boundaries. □

This proposition is used as a conservative extension principle. It allows richer client-local model computations and richer shared-only decoders, while preserving the shared-state factorization argument.

Shared linear solves illustrate the treatment of restricted domains. For $A : \mathsf{Sh}((p,p))$ and $b : \mathsf{Sh}((p))$, define the shared-only extension

$$\mathrm{Solve}_0(A,b) := \begin{cases} \mathrm{Solve}(A,b), & A \text{ is nonsingular}, \\ \mathbf{0}, & A \text{ is singular}. \end{cases}$$

It has output type $\mathsf{Sh}((p))$, and its ordinary interpretation uses the same two branches, so it satisfies the proposition's setup. It is measurable because the nonsingular matrices form an open set and inversion is continuous on that set. Using this extension in a decoder whose only restricted operation is a linear solve gives a decoder defined on the full product state space. The update agrees with the original solve on input and state sequences for which every required matrix is nonsingular. This convention adds no new shared-output boundary.

The operations $\mathrm{ScaleRec}$ and Pred are defined before use in Appendix J. They satisfy the proposition's premises. For aligned inputs with local sample counts n_c, write $b_c = \sum_{d<c} n_d$ for the offset in their virtual global tensors. Then

$$\begin{aligned} \overline{\mathrm{ScaleRec}(U,V)}[b_c + a, j] &= U^{(c)}[a] V^{(c)}[a,j] \\ &= \overline{U}[b_c + a]\, \overline{V}[b_c + a, j], \\ \overline{\mathrm{Pred}(V,\theta)}[b_c + a] &= \sum_{j=1}^{p} \overline{V}[b_c + a, j]\theta_j. \end{aligned}$$

These are the ordinary column-scaled product and row-wise dot product. Their local maps depend on the local shapes and shared parameter, and use no other client's input. When $n_c = 0$, the corresponding index set is empty and the same identities hold. Coordinate projections and the sample-wise outer product satisfy the ordinary-agreement assumption by the same entrywise argument. None of these operations returns a shared tensor from a federated input.

L Implementation and empirical evaluation

FeTA (Federated Tensor Algebra) provides a Python implementation for the two sorts of Section 2. FeTA represents federated and shared values by the wrapper classes `FederatedTensor` and `SharedTensor`. A user writes tensor operations, and elimination of the tracked federated axis identifies an aggregation boundary. Theorem 1 and its extraction construction describe the corresponding mathematical encoder, merge, and decoder.

A controller/executor interface connects the tensor program to backend orchestration, including an NVIDIA FLARE adapter. Runtime wrappers check placement, local shapes, alignment of federated operands, and operation effects when tensor values are available. Two federated operands in an elementwise operation must have matching local shapes. A shared operand must satisfy the broadcast condition of Definition 31. A federated-federated matrix product may expose shared state only through the permitted contraction of the two tracked axes. Rejection of an invalid operation precedes the collective associated with that operation; earlier operations in the program may already have communicated.

An alternative aggregation backend must satisfy the required tensor interface and its own numerical and security assumptions. A secure merge requires a protocol and threat model; a differentially private mechanism also requires sensitivity control and randomization. The privacy conditions in Appendix E therefore remain separate from the choice of execution backend.

L.1 Covariance example and communication dependencies

Listing 1: FeTA covariance implementation for aligned one-dimensional federated arrays with total sample count $N > 0$, using population means.

```
def cov(fed_x: FederatedTensor, fed_y: FederatedTensor):
    """Population covariance between aligned arrays."""
    mean_x = fed_x.mean()          # aggregation over samples
    mean_y = fed_y.mean()          # aggregation over samples
    return ((fed_x - mean_x) * (fed_y - mean_y)).mean()
```

Task	FeTA	Flare	Flower	Ray	Syft	TFF
K-Means	48	147	149	65	56	55
LR	38	124	135	65	59	54
Hist	35	119	106	40	40	36
Stats	42	138	127	56	90	52
GLMM	134	272	288	155	139	148

Table 2: Reported LLOC for the five tasks. Lower values indicate less counted source code under the evaluation's counting convention.

Task	FeTA	Flare	Flower	Ray	Syft	TFF
K-Means	46.64	31.38	26.42	42.22	44.14	43.80
LR	48.43	33.08	28.94	39.87	41.44	41.76
Hist	48.96	32.99	33.62	46.20	46.33	47.20
Stats	44.98	28.14	28.31	37.05	29.02	37.76
GLMM	27.40	17.02	14.60	23.71	26.49	24.22

Table 3: Reported MI for the same implementations. Higher scores indicate greater estimated code maintainability under this metric.

Let $x^{(c)}, y^{(c)} \in \mathbb{R}^{n_c}$ contain paired samples, and let $N = \sum_c n_c > 0$. The semantics of each `mean` is the merge of local sums and counts, followed by division by the total count. Thus, clients with unequal n_c receive their sample weights, and an empty client contributes $(0, 0)$. With $\bar{x} = N^{-1} \sum_{c,a} x_a^{(c)}$ and $\bar{y} = N^{-1} \sum_{c,a} y_a^{(c)}$, the displayed expression computes

$$\frac{1}{N} \sum_c \sum_{a=1}^{n_c} (x_a^{(c)} - \bar{x})(y_a^{(c)} - \bar{y}),$$

the population covariance, corresponding to `ddof=0`. This example has no value specified when $N = 0$.

The source contains three reductions. The first two are independent, and the centered-product reduction depends on their shared results. There are two dependency layers if the first two reductions are batched; an eager backend may execute three collectives. Consequently, this displayed source is not a one-round program under Definition 6. The user code contains no client loop or message-passing routine. Operations over the tracked axis identify the aggregation boundaries.

TFF can express the same algorithm by composing local sum/count maps, federated aggregation of those summaries, broadcast of the resulting means, and a local centered-product map followed by aggregation. FeTA obtains those boundaries from tensor operations. This comparison concerns source organization and typing, and makes no claim of fewer necessary communication phases.

L.2 Reported code-complexity results

The evaluation compares five tasks across FeTA, Flare, Flower, Ray, Syft, and TFF: K-Means, logistic regression (LR), histogram computation (Hist), a suite of eleven summary-statistics routines (Stats), and a generalized linear mixed model (GLMM). The summary-statistics suite includes Pearson correlation and standardized mean difference. Each task uses a common centralized reference. Tables 2 and 3 give the logical lines of code (LLOC) and maintainability index (MI) reported for these implementations, computed with Radon.

In this collection, FeTA has the lowest LLOC and highest MI for all five tasks. These metrics are proxies for code complexity, rather than measurements of runtime performance or developer productivity. The reported validation found agreement with centralized references on the tested configurations. This is not a proof of semantic correctness for arbitrary Python programs.

M Complexity of the Shared-State Factorization

Consider a chosen evaluation of the one-round factorization

$$\sigma_{\mathcal{C}}(X) = \psi\left(\bigoplus_{c\in\mathcal{C}} \varphi_{n_c}(X^{(c)})\right),\ m = |\mathcal{C}| \geq 1,\ D = \dim M = \sum_{i=1}^{q}\prod_{j}(\mathbf{s}_i)_j,$$

where an empty shape contributes one coordinate. The dimension D depends on the program's shared-state shapes, independently of m and the local sample counts. This coordinate bound gives no bound on the cost of an arbitrary encoder, merge, or decoder.

Let $T_\varphi(n_c)$, $T_\oplus(D)$, and $T_\psi(D)$ be costs for the chosen encoder, pairwise merge, and decoder. These symbols may denote upper bounds; dependence on tensor values, primitive implementations, and program parameters is suppressed. Initializing a sequential fold with the first client message gives the work decomposition

$$W_{\text{seq}} \leq \sum_{c\in\mathcal{C}} T_\varphi(n_c) + (m-1)T_\oplus(D) + T_\psi(D). \tag{2}$$

The same expression is the work sum when the cost symbols denote stage costs and each merge call has the displayed cost. A fold initialized at the monoid identity instead uses m merge calls. Theorem 1 supplies the factorization.

Merge storage and depth. Suppose a merge can be evaluated with $O(D)$ workspace. A streaming fold then uses $O(D)$ auxiliary merge storage, independently of m. This bound excludes encoder and decoder workspace, retained data, and simultaneously buffered messages. Associativity permits a balanced tree with $\lceil \log_2 m \rceil$ merge stages. If encoders execute concurrently, merges at each stage execute concurrently, and each merge has cost at most $T_\oplus(D)$, the arithmetic critical path is at most

$$\max_c T_\varphi(n_c) + \lceil \log_2 m \rceil T_\oplus(D) + T_\psi(D).$$

Transport latency, bandwidth, scheduling, and available processors require a separate model before this expression becomes a wall-clock bound.

Dense messages and elementwise merges. If every client transmits one dense, uncompressed element of M, the uplink contains $\Theta(mD)$ real coordinates. If the merge processes each coordinate in constant time, the $m-1$ pairwise merges require $\Theta((m-1)D)$ work, with zero merge work for $m = 1$. These bounds refer to that communication scheme and merge implementation. They are not minimum communication bounds. A bit count requires a representation and precision model. The expression excludes broadcasts of shared parameters and other protocol traffic. Decoder work is the separate term $T_\psi(D)$.

Learning examples. A summed gradient in $\mathbb{R}^p$ has $D = p$. An unknown sample count adds a scalar if it is not already shared. Momentum and Adam maintain $O(p)$ shared server state in addition to the gradient message. For the dense curvature update of Corollary 3, a message containing $g \in \mathbb{R}^p$ and $C \in \mathbb{R}^{p\times p}$ has $D = p + p^2$. A conventional dense linear solve may require $O(p^3)$ arithmetic, so a small dependence on client count does not imply a small decoder cost. Diagonal, block, low-rank, or sketch representations can reduce transmitted state only when supported by the corresponding mergeable expressions. Without the required matrix structure, such representations change or approximate the update. A rank-r label does not provide a mergeable $O(pr)$ representation.

Round-wise use. At round t, let $D_t = \sum_{i=1}^{q_t}\prod_j(\mathbf{s}_{t,i})_j$, with current shared state treated as an encoder parameter. The same decomposition applies with D_t, the participating set $\mathcal{C}_t$, and the costs of that round. Local work may depend on local sample count and the number of local optimizer steps. Message dimension and persistent shared memory are separate quantities; neither bounds local work. Virtual-global agreement concerns the result of evaluation and does not assert equality of federated and centralized running times.

Participation, ordering, and topology. For a sampled finite nonempty set $\mathcal{C}_t$, condition on that set and the current shared state, then apply the deterministic factorization to the participating federation. Sampling provides external randomness. This does not identify a sampled gradient with the full-data gradient or establish unbiasedness or convergence. In the sum/count and weighted-averaging examples, empty local data contribute the merge identity. The decoder uses its declared convention

when the total count is zero. A round with no participating clients can be assigned a separate skip-update convention.

Associativity and commutativity permit tree or hierarchical evaluation of the same merge without requiring a physical star. Equality under reassociation uses the stipulated real-valued operations. Floating-point results may differ. Moreover, permuting clients leaves $\psi(\bigoplus_c \varphi_{n_c}(X^{(c)}))$ unchanged. A sequential client update whose output changes with client order is therefore outside the language's scope. Asynchronous shared models, updates that depend on the order of arrival, and peer states that differ across clients are not covered by this scheduling argument. Persistent private state requires the additional semantics discussed in Appendix G.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes].

   Justification: The abstract and introduction accurately describe the paper's contributions: a typed federated tensor language, shared-state factorization results for one-round and iterative programs, and a differentiable fragment for learning updates. They also state the scope of the framework as computations whose communication passes through fixed-dimensional shared state.

2. **Limitations**

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

   Answer: [Yes].

   Justification: The paper explicitly discusses limitations in the introduction and discussion, including that the base language does not cover all federated algorithms, persistent private client state, exact kernel methods with state growing in the number of records, or formal privacy guarantees without additional cryptographic or differential-privacy execution structure.

3. **Theory assumptions and proofs**

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

   Answer: [Yes].

   Justification: The paper states the assumptions for its semantic propositions, shared-state factorization theorems, iterative-program theorem, conservative-extension result, and differentiable-learning corollaries. The main text provides the formal definitions and theorem statements, and the proofs are provided in the appendices.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper?

   Answer: [N/A].

   Justification: The paper does not report empirical results that support the main claims. The main contributions are theoretical and semantic, while the examples are mathematical instantiations of the proposed language.

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

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

   Answer: [N/A].

   Justification: The paper does not introduce a dataset, benchmark, released model, or experimental codebase as part of its main contribution. Its central results are formal definitions, propositions, theorems, corollaries, and proofs.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details necessary to understand the results?

   Answer: [N/A].

   Justification: The paper does not contain training or test experiments. The learning sections analyze expressibility of gradients, optimizer updates, and curvature-block updates in the typed language rather than empirical model performance.

7. **Experiment statistical significance**

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

Answer: [N/A].

Justification: The paper does not report stochastic empirical comparisons, benchmark experiments, or performance measurements for which error bars or statistical significance tests would be applicable.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources needed to reproduce the experiments?

   Answer: [N/A].

   Justification: The main contribution is theoretical and does not depend on compute-intensive experiments. The paper mentions implementation-oriented future work, but no experimental claims depend on compute resources.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics?

   Answer: [Yes].

   Justification: The work is a theoretical study of typed federated tensor computation. It does not involve human subjects, scraped data, deployed user-facing systems, or release of high-risk models, and it explicitly distinguishes semantic exposure discipline from formal cryptographic or differential-privacy guarantees.

10. **Broader impacts**

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

    Answer: [Yes].

    Justification: The paper discusses federated learning and analytics as tools for decentralized computation over data that should not be centralized, which is a positive privacy-related motivation. It also states that the language alone does not provide formal privacy, legal-compliance, cryptographic, or differential-privacy guarantees, making clear the risks of overinterpreting the semantic guarantees.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse?

    Answer: [N/A].

    Justification: The paper does not release datasets, pretrained models, generative systems, or other assets with a high risk of misuse.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets used in the paper properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A].

    Justification: The paper does not use existing datasets, pretrained models, or third-party code assets as part of the main contribution. Prior scientific work and systems are cited in the bibliography.

13. **New assets**

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

    Answer: [N/A].

    Justification: The paper does not release new datasets, models, benchmarks, or software artifacts.

14. **Crowdsourcing and research with human subjects**

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

Answer: [N/A].

Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board approvals were obtained?

Answer: [N/A].

Justification: The paper does not involve human subjects, so IRB or equivalent approval is not applicable.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research?

Answer: [Yes].

Justification: The authors used LLM-based tools for writing assistance, editing, organization of exposition, and interactive brainstorming or clarification of technical concepts. The LLMs were not used as autonomous sources of scientific claims or mathematical validity. All definitions, theorem statements, proofs, and claims were reviewed, revised, and verified by the authors, who take full responsibility for the content of the paper.