Doc. No. P4025R2
Date: 2026-05-10
Audience: SG19
Author: Michael Wong
Reply to: fraggamuffin@gmail.com
Contributor: Phil Ratzloff

Revision History

R2

- I made a mistake and used AI to generate just the revision history and so have updated that. The text since R0 is mine.

R1

- Updated **Priority 1 (`std::data_frame`)** to include standard data ingestion requirements for **CSV** and **JSON** formats.
- Added the "Ingestion Imperative" section to formalize strategic alignment with **SG16 (Unicode/Text)** for text parsing.
- Defined the required "Output Shape" for standard parsers, specifying the need to directly populate `std::data_frame` columns and `std::mdarray` tensors

R0

- Initial paper of the SG19 strategic direction after discussion in SG19.
- Started with the immediate-term Data Structures targeted for C++29, including `std::data_frame`, `std::mdarray`, `std::graph`, and `std::statistics`.
- Outlined the medium/long-term Execution Models targeted for C++32, including Runtime Generation (JIT/AutoDiff), Ultra-Low Precision Arithmetic (FP8/int4), and Metadata-Aware Smart Tensors.

The SG19 Priority List for C++29/32

C++ is the engine of AI, but lacks key components required to prevent fragmentation. After many discussions in a non-vendor specific manner, SG19 identified the following features as **Strategic Priorities** for the C++29/32 timeframe so that C++ remains the engine of AI. Work on these features in SG19 (Machine Learning), SG14 (Low Latency), and SG6 (Numerics) started in SG19/14 in November 2018(San Diego), continues and is critical.

This direction is about what is immediately doable for C++29 A subsequent direction will be for beyond C++29.

Part I: Immediate Priorities for C++29 (The "Nouns" & Data Structures) These proposals focus on fixing the data structures required for Interop and Ingestion.

Priority 1: The Data Science Foundation (`std::data_frame`) & Standard Ingestion (CSV/JSON)

- **Status:** Critical Gap / Call for Proposal. (Requires alignment with SG16)
    - l such as this example which has application in financial quants
        - This is a similar idea: https://github.com/hosseinmoein/DataFrame
- **Strategic Need:** AI workflows currently default to Python/Pandas because C++ lacks a native Data Frame and frictionless data ingestion. To remain the engine of AI, C++ needs a standard, heterogeneous, column-oriented container (`std::data_frame`) that allows "Zero-Copy" data exchange with the broader ecosystem.
- **Format Scope:** We require robust, standardized ingestion for both **CSV** and **JSON** to support modern and legacy datasets.
- **The "Output Shape":** The critical requirement for SG19 is that the output of these parsers seamlessly bridges the gap between raw text and mathematical structures. Parsers should be designed to directly and efficiently populate the columnar structures of `std::data_frame` or the tensor views of `std::mdarray` without requiring intermediate string allocations or heavy glue code.

Priority 2: The Core Math (`std::linalg` & `std::mdspan`), also add mdarray (SG14/6)

- **Status:** `mdspan` is C++23; `linalg` is in flight (C++26). **`mdarray` is the target for C++29.**
- **Strategic Need:** While `std::mdspan` provides the "view," we lack the owning container. **`std::mdarray`** is required to simplify memory management for tensors. `std::linalg` is required to provide portable, standard linear algebra (BLAS) operations that can dispatch to hardware-optimized backends.

Priority 3: Graph Data Structures (`std::graph`)

- **Status:** In development (SG19).To enter LEWG
- **Strategic Need:** Graph Neural Networks (GNNs) and Retrieval-Augmented Generation (RAG) are dominant AI workloads. `std::graph` provides the standard topology for these applications. We urge the committee to resolve design trade-offs (e.g., concepts vs. virtual hierarchy) to deliver this facility.
- std::graph adds an interesting twist because it's a range-of-ranges, not just a simple homogenous container. It's possible to load edges and vertex values separately, but it might be possible to have multiple types in a CSV file where each row has a column value that identifies a row type that generates different rows, one for vertices and another for edges.

Priority 4: Statistics (`std::statistics`)

- **Status:** In development (SG19).In LEWG.
- **Strategic Need:** Fundamental building blocks (mean, variance, skewness) are required for almost all AI normalization layers and stochastic gradient descent. This should be a high priority for standardization.

Part II: Strategic Directions for beyond C++29 (The "Verbs" & Execution Model)

Future proposals must prevent C++ from becoming just a "dumb backend" for Python. These focus on **Execution, Quantization, and Code Generation.**

1. The "JAX" Killer: Automatic Differentiation & JIT

- Goal:Auto differentiation similar to a previous SG19 proposal but it requires reflection support which we have in C++26 (though there may be gaps). Move beyond Ahead-of-Time (AOT) dispatch to Runtime Generation.
- Need: Support differentiation which is the fundamental of stochastic gradient descent at compile time. This allows C++ to do what JAX does (compile-time optimization of math) without the Python overhead. Support Computation Graphs that can fuse ASTs of operations to handle Kernel Fusion (e.g., merging `MatMul + ReLU + Bias` to save memory bandwidth).
- Comparison: This is required to compete with Python-driven JIT compilers like OpenAI Triton and PyTorch Inductor which is the next Era of AI.

2. Ultra-Low Precision Arithmetic for modern LLMs & Quantization

- Goal: C++23 has float15_t and bfloat15_t. But we need native support for `std::float8_t` (FP8) and `std::int4_t` for TPUs, and "nibble" types for weight compression.
- Need: Without standard ultra low-precision types, `std::linalg` and `std::simd` cannot be used for modern AI inference. We also need a Quantization Schema for `mdspan` that automatically handles the de-quantization math such as `scale` and `zero_point` logic on access.

3. Smart Tensors (Metadata-Awareness)

- Goal: Metadata-Aware `mdspan` (Named Tensors).
- Need: Errors from mixing up "Batch" and "Channel" dimensions are the #1 source of bugs in implementing Transformers. This is a low-cost, high-value safety feature. By allowing users to tag dimensions with compile-time strings or types, you eliminate an entire class of runtime errors ("Safety by Construction") without runtime overhead. This fits the **Safety** narrative Priorities for C++29 applied specifically to the AI domain. Closing the "Python Gap" by allowing tensors to carry semantic tags (e.g., "Batch", "Channel") to prevent shape-mismatch errors.

There are other potential future features which SG19 discussions will continue to surface.