Building a Rust-Based Tensor ~~Network~~ Ecosystem

with Agentic AI Engineering

Hiroshi Shinaoka

Saitama University, Japan

Advertisement: postdoc opportunities in Japan

JSPS Postdoctoral Fellowships for Research in Japan:

Standard

- 12 to 24 months
- PhD within 6 years
- Deadline: **Aug 28, 2026**

Short-term (PE)

- 1 to 12 months
- US, Canada, Europe
- Deadline: **Sep 25, 2026**

Talk to me this week.

Host institution submits the application. JPY 362,000 per month, airfare covered. www.jsps.go.jp

One substrate for tensor workflows

From QMC, *ab-initio* calculations, tensor networks, to neural networks

- **Needed**: One common, high-performance substrate
- Fragmented across Python, Julia, C++, Fortran...
- **Agentic AI engineering** → one unified, trustworthy **Rust** stack.

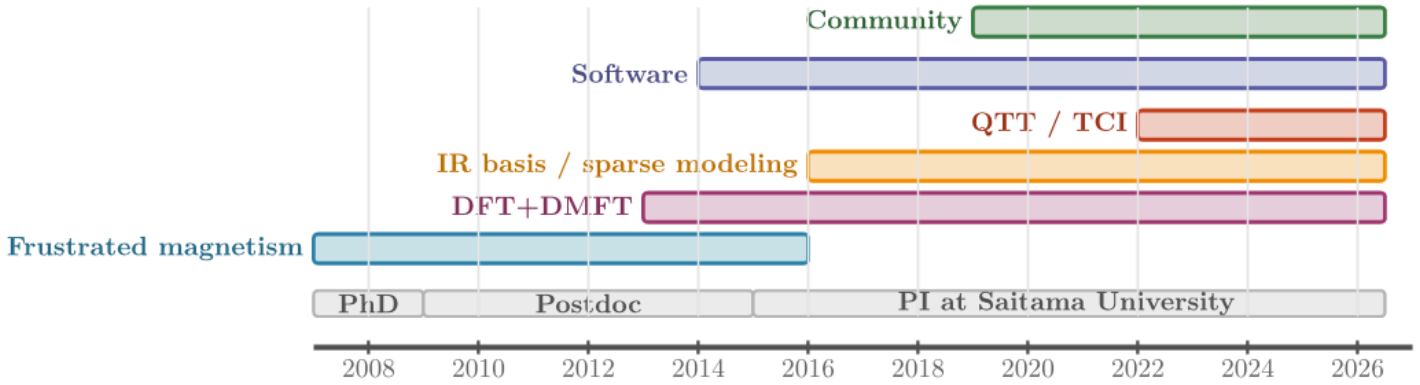
Outline

1. Motivation: compressing quantum many-body data (IR, QTT/TCI)
2. The tensor4all ecosystem: collaboration and libraries
3. The bet: Rust + agentic AI
4. The new stack: tenferro-rs + tensor4all-rs
5. How we build it: agentic AI engineering
6. Education and outlook

Motivation

I build instruments for computational condensed matter

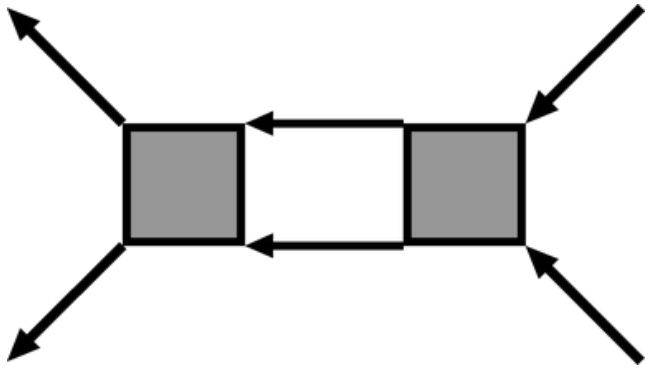
- PhD (Tokyo) → postdoc (Switzerland) → PI at Saitama, since 2015.
- Quantum Monte Carlo, DFT+DMFT for correlated materials.
- Sparse modeling, the **IR basis** for imaginary-time data.
- **QTT / TCI** for high-dimensional functions.



Why compression became necessary

- Advanced many-body theory: DFT+DMFT, GW, functional RG.
- These act on **response / correlation functions**: high-dimensional in space-time.
- A four-point function: $\sim 10^{12}$ grid points (10^3 each axis)

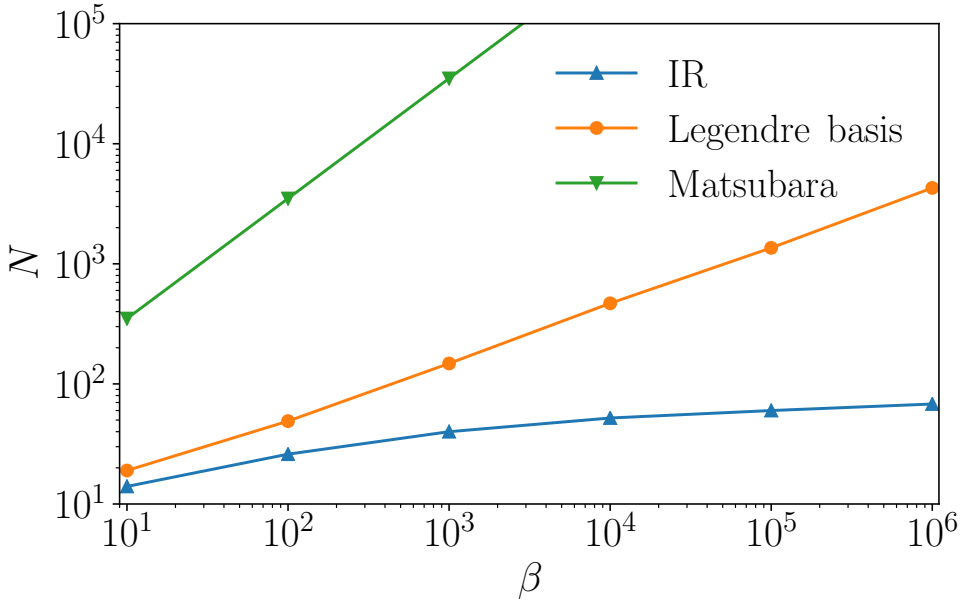
High-dimensional → dimensionality reduction is essential.



Compressing quantum many-body data

IR basis: result first

Naive Matsubara grid: $N \sim O(\beta)$, intractable at low T.



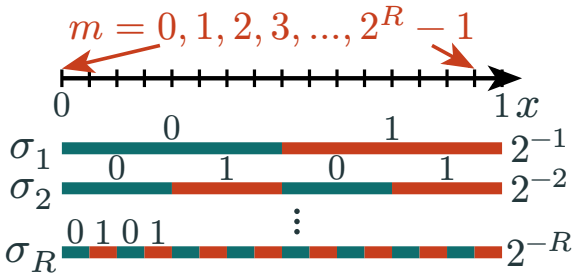
- Gap: temperature $O(1)$ K vs bands $O(10)$ eV $\rightarrow \beta\omega_{\max} \geq 10^4$.
- IR basis: $\boldsymbol{N \sim O(\log \beta)}$.
- **sparse-ir**: GW, DMFT, Eliashberg.
- Python / Julia / Fortran / C++ / Rust.

200+ citations: HS et al., PRB 96, 035147 (2017); Li et al., PRB 101, 035144 (2020)

QTT: tensorize the grid

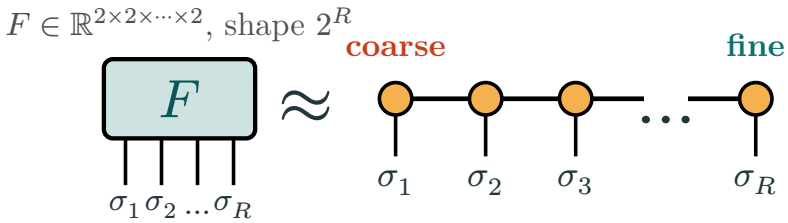
Tensorize the grid

- Grid $N = 2^R$, binary index $\sigma_1 ... \sigma_R$
- Reshape $\rightarrow$ R-way tensor F



Use scale separation

- Scales separate $\rightarrow$ F factorizes as a TT
- Bond $\chi \rightarrow$ **$O(R\chi^2)$ params**

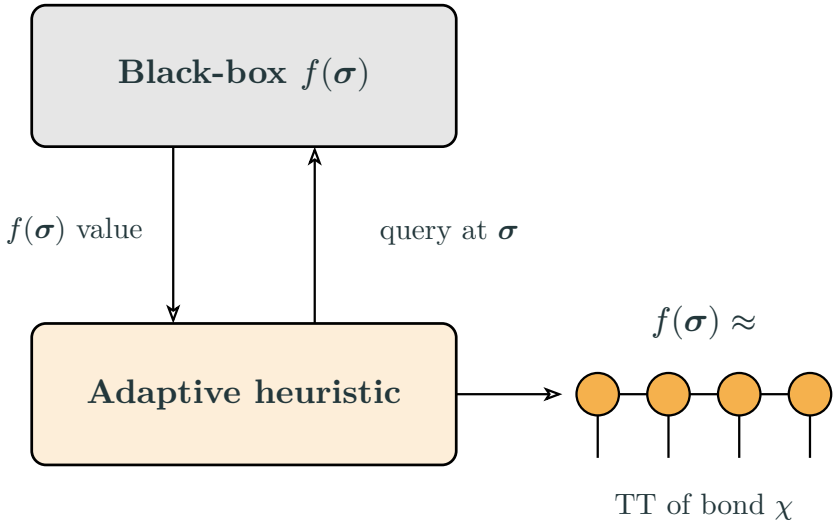


QTT stores and computes: convolutions, contractions, linear solves in compressed form.

Oseledets, Dokl. Math. 80, 653 (2009); Khoromskij, Constr. Approx. 34, 257 (2011); quantum many-body: **HS** et al., PRX 13, 021015 (2023)

TCI: learn a TT by querying a black box

- Oracle $f(\sigma_1, ..., \sigma_L)$ + initial pivots
- **Adaptive** queries → learns a TT
- Cost: $O(\chi^2 L)$ calls, only QR/LU (no SVD, no gradient descent)
- Output: near-optimal bond χ



For the algorithm, see Yuriel's talk from yesterday afternoon.

Oseledets 2011; Núñez Fernández et al., PRX 12, 041018 (2022)

Recent highlights

- **Multi-orbital impurity solver**: A_3C_{60}, 3 orbitals + phonons (Ishida, PhD)
- **Long-time nonequilibrium** GW: divide-and-conquer (Inayoshi, postdoc)
- **Nonequilibrium** GW quench: **6 TB → 1 GB** in QTT form (Środa + HS)
- **Weak-coupling free energy**: TCI-compressed diagrammatics (Matsuura)
- **Two-particle vertex**: parquet/BSE solved entirely in QTT (TU Wien)

Ishida et al., PRL 135, 046502 (2025); Inayoshi et al., SciPost Phys. 20, 077 (2026); Środa et al., PRL 135, 226501 (2025); Matsuura et al., PRB 111, 155150 (2025); Rohshap et al., PRR 7, 023087 (2025)

tensor4all ecosystem

tensor4all: the collaboration

Goal: interdisciplinary community for physics and computational/applied math around tensor learning methods.

- **Funded in 2025**: CEA Grenoble, Flatiron Institute, LMU Munich, Saitama U.
- **Monthly online meetings**: developer meetings, talks by physicists and mathematicians.
- **Workshops**: Vienna 2024, Grenoble 2025.

`https://tensor4all.org/`

The libraries that proved QTT/TCI

C++

- xfac: C++ core, Python bindings.
- CEA Grenoble…

Julia

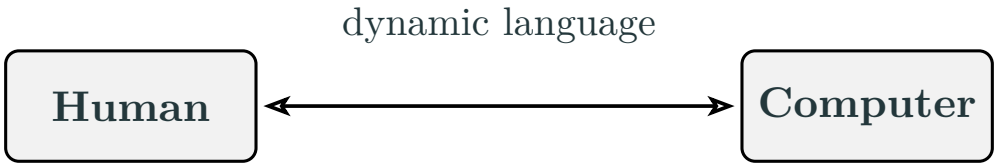
- TensorCrossInterpolation.jl, QuanticsGrids.jl, QuanticsTCI.jl.
- LMU Munich, TU Wien, Saitama U…

Every result shown above came from these libraries.

Núñez Fernández, Ritter, …, HS, Waintal, SciPost Phys. 18, 104 (2025)

Workflow: before and in the agentic AI era

Before: Julia, an expressive language with a REPL.



Now: I chat with an AI agent; it writes and runs the code.



Is Julia still the right choice for that era?

Challenges in building a reliable and maintainable codebase

- Long compilation time
- Unexpected dynamic allocation
- Reusability of code from different languages
- Autodiff is not stable.
- Small user community

Julia was a great choice *before* agentic AI era, and I still like it. But, it is not the right choice for that era.

Now: Rust + agentic AI

Why Rust: language and ecosystem

- **Ownership and static types**: memory safety, no aliasing or data races, all at compile time.
- **Cargo**: integrated build, dependency resolution, testing, benchmarking, parallel compilation.
- Ecosystem covers the basics and grows fast: faer (linear algebra), ndarray, Burn (deep learning), LAPACK/BLAS/HDF5/MPI wrappers.[1]
- Trade-off: a steep learning curve, now absorbed by the AI.

[1] In 2026 NVIDIA released cuda-oxide, an experimental Rust compiler for CUDA kernels.

Why Rust now: the reason inverted

Before agentic AI era: CEA Grenoble colleagues recommended Rust, but I judged it premature.

Target flipped: **cost of writing → cost of verifying**.

Still **missing**: complete tensor library with autodiff and GPU support.
This is what we build.

My Rust journey with agentic AI

October 2025: ported sparse-ir to Rust

- About 2 months, roughly 40k lines of code
- Zero hand-written Rust
- Python / Julia / Fortran bindings, integrated into EPW (*ab initio* electron-phonon)

January 2026: decided to port the whole tensor4all ecosystem to Rust

- tensor4all-rs: 353 commits in 2 months, +61,486 lines in the first 2 weeks

But how can I verify code I did not write?

The new Rust stack

Layered architecture

Tensor4all.jl	14k LOC	Julia interface, ITensors compatible
tensor4all-rs	144k LOC	TreeTN / QTT / TCI / MSID
tenferro-rs	83k LOC	tensor engine: autodiff, CPU / GPU

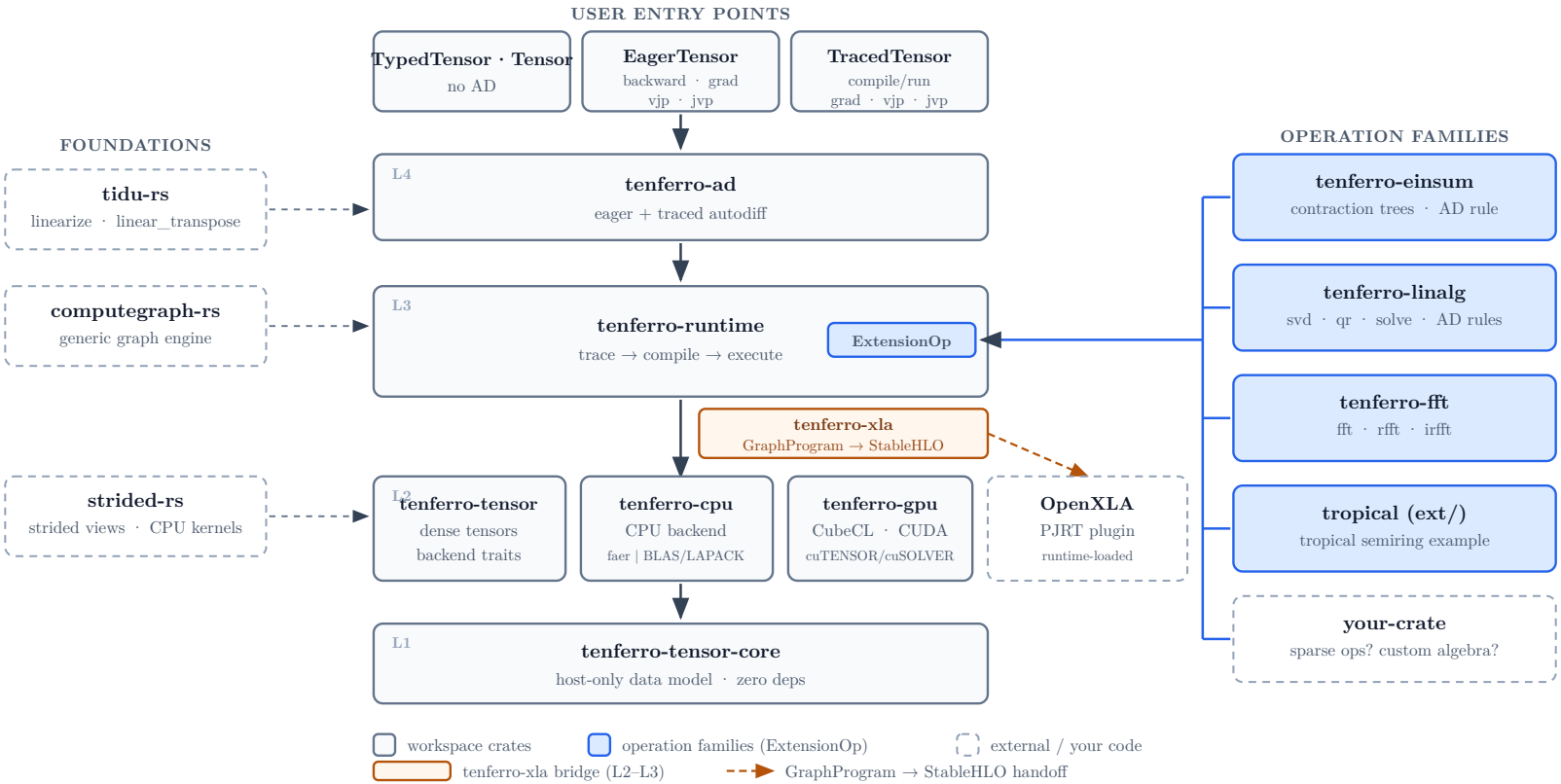
tenferro-rs: a general-purpose tensor library

tenferro = tensor + ferro (iron flavor)

Co-designed with Jin-Guo Liu (HKUST-GZ) and contributions from S. Terasaki and R. Watanabe.

- **Pure Rust**: dense tensors, einsum, FFT, linear algebra.
- **Autodiff** (optional): `vjp` / `jvp` in eager (PyTorch-style) and traced (JAX-style) modes, high-order derivatives with complex-number support.
- **CPU and GPU** from one codebase.

Inside tenferro-rs: layered crate architecture



einsum, linalg, and FFT are not built-ins — they register through the same public ExtensionOp interface available to any crate.

tenferro-rs: performance

- Performance **on par with JAX and PyTorch**
- **tenferro-benchmark**: an external repo measures it continuously.
- **GPU with full CPU feature parity**: same ops and AD on both backends.
- **OpenXLA** backend support (performance matching JAX)

A physics-grade tensor engine, at machine-learning-framework speed.

Benchmarks: github.com/tensor4all/tenferro-benchmark

Sample: a physics-informed net for the KdV equation

Train a network $u_\theta(x, t)$ to satisfy

$$u_t + 6uu_x + u_{xxx} = 0$$

```
// coordinate derivatives: forward mode, jvp (inputs are low-dim)
let u_x   = u.jvp(x, &ones)?;
let u_xxx = u_x.jvp(x, &ones)?.jvp(x, &ones)?;      // 3rd order in x
let residual = u_t.add(&sixuux)?.add(&u_xxx)?;     // u_t + 6 u u_x + u_xxx

// gradient of the scalar loss over all parameters: reverse mode, grad
let grads = params.iter().map(|p| loss.grad(p)).collect()?;
```

Forward `jvp` for the few coordinate derivatives, reverse `grad` for the many parameters.

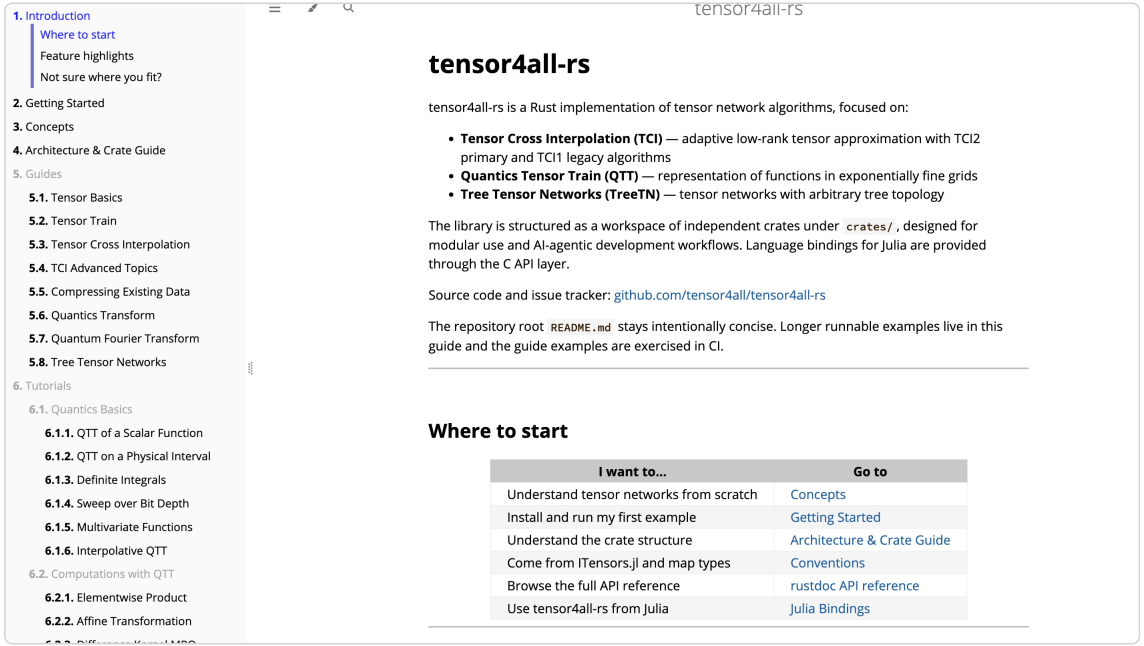
Tutorial: tensor4all.org/tenferro-rs (KdV PINN sample)

tensor4all-rs: the methods stack

- **TreeTN**: tree tensor networks, arbitrary topology, canonicalization + truncation + DMRG + TDVP…
- **QTT**: continuous functions on an exponentially fine grid, polynomial cost.
- **TCI**: tensor-train surrogate of a black box, from few samples.
- **MSID**: multiscale interpolative decomposition, many scales in one pass.

Never reimplement TreeTN, QTT, TCI, or MSID from scratch.

tensor4all-rs: documented and tutorial-driven



- Guides for TCI, QTT, TreeTN.
- Tutorials, from scalar QTT to computations in compressed form.
- Julia bindings via a C API.

Tutorials by **Selina Dirnböck** (TU Wien).

tensor4all.org/tensor4all-rs

Next: How do we verify something this large?

How do we verify something this large?

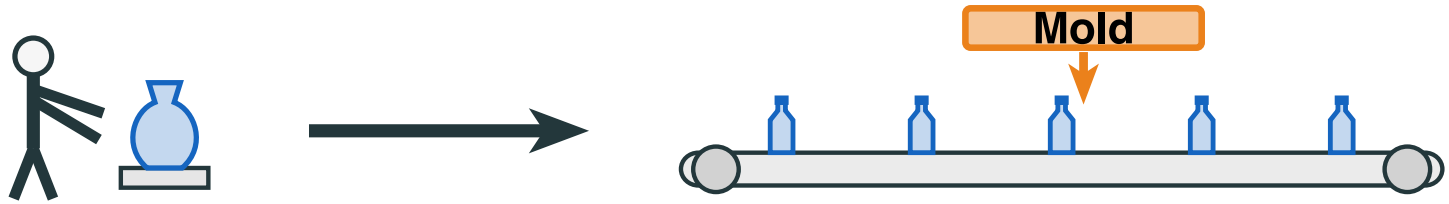
- Over **200k lines** in a half year (tenferro-rs + tensor4all-rs)
- Julia interface, C API, and Rust core.
- Almost none of that Rust was hand-written.

How do we build, and verify the large codebase?

The answer is agentic AI engineering.

How we build it: agentic AI engineering

The industrial revolution of coding: from craft to factory



Humans crafting one piece at a time

= hand coding

Mass production from a mold (fast, uniform)

= ideal agentic coding

The same shift is happening now. We are still figuring out how to realize ideal agentic coding.

A common criticism: “AI slop”

Critiques of agentic-coding output:

- **Loss of craft**: tools rejected even if they worked (Sinclair Target).
- **Maintenance breaks down**: curl shut its bug bounty, buried in submissions.
- **Shared resources degrade**: a tragedy of the commons for reviewers (Baltes et al.).

AI slop = mass generation without a source of truth.

Readable source is not inspectable implementation

Readable and **inspectable** are different.

Even code that **looks** close to the math hides its real behavior:

```
b = a          # b and a may point to the same array (alias)
a .+= c        # if updated in-place, b changes too (side effect)
```

Aliasing, mutation, allocation are invisible from how a line looks.

The cost of verifying: a static, strict language makes the implementation machine-inspectable.

Supporting a huge codebase with three pillars

Line-by-line review is impossible. Instead of a human standing guard, three pillars hold it up.

Pillar 1
tests

unit + integration, parallel, fast feedback

Pillar 2
oracle / benchmark

external standards, kept in a repo the AI cannot edit

Pillar 3
source of truth

rules that grow from every failure

Pillar 2 is public: tensor-ad-oracles (AD oracles) · tenferro-benchmark

Pillar 1 leans on Rust: crates compile in parallel and tests run in parallel by default, so the whole suite finishes in minutes and every AI rewrite gets instant green/red.

Next: Pillar 2 in action: AD benchmarks vs PyTorch and JAX

Pillar 2 in action: AD benchmarks vs PyTorch and JAX

Reverse-mode (VJP) gradients of linalg ops (trace-mode AD). 512×512, f64, 1 thread, CPU. Median ms, lower is better.

operation	**tenferro-rs**	PyTorch	JAX (XLA)
eigh	**79.1**	64.5	109.8
LU	**43.4**	49.4	57.0
QR	**60.2**	64.7	73.7
solve	**6.9**	11.8	15.8
SVD	**140.2**	142.1	198.9

Oracle + continuous benchmark (Pillar 2): github.com/tensor4all/tenferro-benchmark

What agentic coding changes for teaching

Objections from collaborators, and my answers:

	Concern	**My answer**
① Library ≠ app	Apps have no reference answer.	Still checkable: symmetries, limits, conservation laws.
② Learning stage	A senior's workflow is premature for juniors.	Separate by stage: basics by hand, agentic at research level.
③ Julia readability	Notebooks let you eye-check the math.	Eye checks are unreliable; have the AI emit pseudocode to reconcile.

A pipeline from practice into education, not a defense of past habits.

Join us: bridge tensor networks, quantum field theories and neural networks

Users, co-development, and feedbacks are all welcome.

Come to Japan: **JSPS fellowships**, deadlines **August 28** and **September 25**, 2026 (slide 2).

`https://tensor4all.org/`

Closing

tenferro-rs and tensor4all-rs: one verified Rust substrate for neural and tensor networks.

Built by a tiny team, because the human's job moved from writing to proposing, designing, and verifying.

Still evolving: the rules, the stack, and the curriculum all keep growing.