u4lib: An AI-Ready Computational Framework for U(4) Symmetry Algebra, Wigner Coupling, and Racah Recoupling Coefficients

Phong Dang,[1] Daniel Langr,[2] Tomas Dytrych,[3,1] and Michela Negro[1]

[1]*Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA 70803, USA*
[2]*Department of Computer Systems, Faculty of Information Technology, Czech Technical University in Prague, Prague 16000, Czech Republic*
[3]*Nuclear Physics Institute, Czech Academy of Sciences, 250 68 Řež, Czech Republic*

Wigner's supermultiplet symmetry treats nucleon spin and isospin within a common U(4)/SU(4) description. The C++ library `u4lib` supplies the coupling coefficients and algebraic operations needed for calculations in this scheme. Users can construct irreducible representations (irreps), obtain dimensions and Casimir invariants, conjugate states, enumerate the canonical Gelfand-Tsetlin and physical $\mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ bases with inner multiplicities, and decompose tensor products. Matrix-element routines cover the canonical U(4) generators and the physical spin **S**, isospin **T**, and Gamow-Teller-type tensor **U**. Wigner coupling and Racah recoupling coefficients are evaluated by the procedure of Dang *et al.* The implementation builds on `u3lib`, with a header-only C++ interface and corresponding Python bindings. A catalog of agent skills assists an AI agent in preparing, executing, and checking calculations through natural-language requests.

PROGRAM SUMMARY

Program title: `u4lib`
Licensing provisions: MIT License.
Code repository: https://github.com/Deep-SANE/u4lib
Programming language: C++ (C++20), with Python bindings via pybind11.
External libraries: `u3lib` is a prerequisite, through which core utilities and other dependencies—namely `Eigen3` for QR-based null space solver, `Boost` for hash tables, `OpenMP` for multi-threading, `fmt` for formatting output, `libwignernj` for SU(2) coupling and recoupling coefficients, `KraNS` for sparse null-space solvers on large matrices (with a dense-QR fallback), and `pybind11` for Python binding—are inherited.
Nature of problem: Efficient, accurate, and reproducible computation of Wigner reduced coupling coefficients for the two subgroup chains $\mathrm{U}(4) \supset \mathrm{U}(3)$ and $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$, Racah recoupling coefficients (6-U(4) and 9-U(4) coefficients), and the associated representation-theoretic quantities (dimensions, Casimir invariants, multiplicities, basis enumeration, generator matrix elements, and basis transformations) necessitated by symmetry-adapted nuclear-structure calculations and similar applications.
Solution method: The reduced coupling and recoupling coefficients are built from the highest-weight canonical $\mathrm{U}(4) \supset \mathrm{U}(3) \supset \mathrm{U}(2) \supset \mathrm{U}(1)$ full coupling coefficients, obtained as the orthonormal null space of the canonical raising generators, and propagated by the lowering counterparts as shown in [1]; the physical $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ coefficients follow from the physical-to-canonical basis transformation, which is also determined through the null space of the spin raising S_+ and isospin raising T_+ operators [1].

I. INTRODUCTION

Wigner's supermultiplet symmetry was proposed in 1937 as the overarching scheme that unifies the spin and isospin degrees of freedom of atomic nuclei [2]. It is not an exact symmetry of the strong interaction, but rather emerges in the limit of many colors in QCD [3] and at a nuclear-force resolution of roughly 500 MeV [4, 5]. In spite of its broad range of applications, from nuclear masses [6–8], β-systematics [9–11], and Gamow-Teller resonances [12], to more recent lattice simulations [13] and algebraic models that mitigate symmetry breaking in heavy nuclei [14, 15], there has been limited effort to probe and leverage the supermultiplet symmetry in first-principles shell-model calculations as compared to its spatial SU(3) [16–18] and symplectic $\mathrm{Sp}(3,\mathrm{R})$ [19, 20] counterparts.

This underdevelopment stems from the fact that working in the physical spin-isospin $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ basis requires U(4)/SU(4) coupling and recoupling coefficients. While the computation of these coefficients was studied earlier by Hecht and Pang [21], by Draayer [22], by Partensky and Maguin [23] and by Rowe and Repka [24], no method produced scalable software for large-scale nuclear structure calculations. A recent Python package [25] that automates the evaluation of $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ branching multiplicity has been advanced. Most recently, building on efforts by Pan *et al.* to develop the physical basis and elementary physical coupling coefficients [26–28] along with previous work on canonical coupling coefficients for SU(4) [29] and the more general SU(N) [30], Dang *et*

al. have developed a generic algorithm [1] that supplies any and all physical $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ coupling and recoupling coefficients, propelling the first *ab initio* shell-model calculations using Wigner's symmetry [31].

In a nutshell, the new algorithm leverages a property of the group generators: the highest-weight state of an irrep is annihilated by the raising operators, which impose a system of homogeneous linear equations that can be solved by a numerical null space solver to yield the full coupling coefficients for the highest-weight state as an orthonormal set of null space vectors. For the large sparse matrices arising in this approach, a dedicated iterative null-space solver, now known as `KraNS`, was developed [32]. All other coefficients then naturally follow through the action of the lowering operators. The same strategy has also been employed to modernize the 1973 Draayer-Akiyama formalism [33, 34] for computing SU(3) coupling coefficients [35], implemented in the modern library `u3lib` we should put u3lib to arXiv first to get a reference here .

This paper presents `u4lib`—a header-only C++ library with Python wrappers and a sibling of `u3lib`—which implements the algorithm demonstrated in Ref. [1] and is part of a broader effort called `DeepSANE` (**Deep**-learning and **S**ymmetry-adapted **A**b-initio **N**uclear modeling: an AI-ready **E**cosystem[1]). To our knowledge, this is the first library capable of computing generic coupling and recoupling coefficients for the supermultiplet symmetry. A calculation herein can begin with either a canonical state in the $\mathrm{U}(4) \supset \mathrm{U}(3) \supset \mathrm{U}(2) \supset \mathrm{U}(1)$ chain or a physical state labeled by spin and isospin quantum numbers in the $\mathrm{SU}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$ chain. The library supplies the transformation between these two bases, state conjugation, generator matrix elements, Kronecker (tensor-product) decompositions, and Casimir invariants alongside the coupling and recoupling routines. Its dependence on `u3lib` reflects the canonical subgroup chain and allows the two libraries to share numerical infrastructure.

Following the broader effort to modernize scientific workflows in the agentic-AI era, we also provide agent-ready domain skills in `u4lib`. With an aim similar to that of the Genesis Mission's agent skill catalog[2] of the U.S. Department of Energy, these skills empower AI assistants to become specialists capable of writing programs, debugging, and facilitating adoption. To demonstrate their usefulness, we assess agents' performance through three coding tasks, measuring whether access to the skill catalog changes correctness, token use, runtime, and API cost.

Note that while our work lies primarily within nuclear physics, the same algebraic structure is also adopted in various fields. In quantum information, the two-qubit unitary group is SU(4), and the geometry of its nonlocal gates follows from a Cartan decomposition of the algebra [36]. In ultracold-atom physics, alkaline-earth atoms realize SU(N)-symmetric magnetism that includes SU(4) [37], while in condensed matter the spin-valley degrees of freedom of graphene form an approximate SU(4) quantum Hall ferromagnet [38]. These applications may use the representation data supplied by `u4lib`, though their physical models and basis conventions must be handled separately—we cordially invite researchers from other fields to contribute to the library to support their respective applications.

The rest of the paper is organized as follows. Sections II and III describe the source layout and C++ reference interface. Section IV covers Python use, and Sec. V explains the skills and reports the coding experiments. Build instructions appear in Sec. VI.

II. LIBRARY STRUCTURE

The headers contain the U(4) implementation and use `u3lib` as a dependency. Examples, bindings, tests, and documentation occupy separate directories:

- `include/`: the library itself (all algorithms live here; user programs include these headers directly):
  - `U4irrep.h`: the irrep classes `su4::irrep`, `u4::irrep`;
  - `U4gelfand.h`, `internal/gelfand/`: the Gelfand state class `u4::gelfand`;
  - `U4basis.h`: basis enumeration (`gelfandBasis`, `physicalBasis`), the inner multiplicity `Zeta`, the `u4::physicalvector` class, and the generator-action predicates;
  - `U4decomposition.h`: the outer multiplicity `Eta` and the tensor-product `decomposition`;
  - `U4U3xU4U3.h`, `U4SU2SU2xU4SU2SU2.h`: the canonical and physical coupling coefficients and their databases;
  - `U4_U6Z6Coeffs.h`, `U4_U9Coeffs.h`: the 6-U(4) and 9-U(4) recoupling coefficients and their databases;
  - `U4SU2SU2_to_U4U3U2U1.h`: the physical-to-canonical transformation;
  - `U4generatorsRME.h`: the physical-generator reduced matrix elements `RME_S`, `RME_T`, `RME_U`;

[1] See the `DeepSANE` GitHub organization: https://github.com/Deep-SANE
[2] See the DOE Genesis Mission skill catalog: https://gitlab.osti.gov/genesis/genesis-skills.

  - `Definitions.h`: the structured coefficient containers;
  - `u4lib.h`: the umbrella header and the `initialize`/`finalize` lifecycle.

- `src/`: small example driver programs (one `.cpp` per task) that call the headers and print results; compiled to `bin/`.
- `python/`: the pybind11 extension (`pyu4lib.cpp` → `u4lib.so`), its build (`make python`, `build.sh`), build notes (`BUILD_NOTES.md`), and a tutorial Jupyter notebook (`tutorial.ipynb`).
- `agents/`: the AI-agent skill catalog: an index (`AGENTS.md`), shared runtime instructions, and one skill per capability.
- `test/`: the regression suite (`test_*.cpp`, the `*_cases.h` case sets, and `reference/` documented values), run by `make test`.
- `doc/`: this manual (LaTeX source).
- `bin/`: build output (not source).

III. CLASSES AND LIBRARY INTERFACE

The interface separates algebraic objects from coefficient databases. Irreps, basis states, and their methods can be used immediately; database queries require the lifecycle calls below. U(4) types use the `u4` namespace and inherit SU(4) functionality from `su4`. Table I identifies the relevant header for each operation.

Header	Entry point	Task
`U4irrep.h`	`dim`, `C1/C2/C3/C4`, `conjugate`	irrep dimension, Casimirs, conjugation
`U4basis.h`	`Zeta`, `gelfandBasis`, `physicalBasis`	inner multiplicity; basis enumeration
`U4decomposition.h`	`Eta`, `decomposition`	outer multiplicity; Kronecker products
`U4U3xU4U3.h`	`wigU4U3db`	U(4)⊃U(3) canonical coupling coefficients
`U4SU2SU2xU4SU2SU2.h`	`wigU4SU2SU2db`	$SU_S(2)\times SU_T(2)$ physical coupling coefficients
`U4_U6Z6Coeffs.h`	`u4u6db`, `u4z6db`	6-U(4) recoupling coefficients
`U4_U9Coeffs.h`	`u4u9db`	9-U(4) recoupling coefficients
`U4SU2SU2_to_U4U3U2U1.h`	`phys_to_cano`	physical-to-canonical transformation
`U4generatorsRME.h`	`RME_S`, `RME_T`, `RME_U`	reduced matrix elements of **S**, **T**, **U**

TABLE I: Main entry points of `u4lib`.

A. Lifecycle

In `u4lib.h`:

- `initialize()` / `initialize(&argc,&argv)`: set up the caches of SU(2) coupling and recoupling coefficients, and the U(3) and U(4) coefficient databases (the cascade descends into `u3lib`);
- `finalize()`: release these resources in reverse order;
- `initialize_thread()` / `finalize_thread()`: per-thread setup and teardown for multithreaded use;
- `clear()`: discard cached coefficients while leaving the databases intact;
- `is_initialized()`: report the current state.

Shared initialization: `u4lib` initializes its `u3lib` dependency automatically. If the application also calls `u3lib` directly, initialize both libraries at startup and defer finalization until all coefficient calculations have finished. Their `libwignernj` backend is shared; finalizing either library while the other still needs it causes runtime errors.

B. Irreps: `su4::irrep` and `u4::irrep`

(`U4irrep.h`.) `su4::irrep` stores an SU(4) irrep (P_1, P_2, P_3); `u4::irrep` derives from it and adds the fourth label n_{44} to fix the total quanta.

Constructors:

- `su4::irrep()`: default (empty) SU(4) irrep.
- `su4::irrep(P1,P2,P3)`: SU(4) irrep from its labels.
- `u4::irrep()`: default (empty) U(4) irrep.
- `u4::irrep(n14,n24,n34,n44)`: U(4) irrep from its four labels.
- `u4::irrep(P1,P2,P3)`: normalized representative.
- `u4::irrep(su4::irrep, A)`: fix the total quanta A.

Label getters:

- `P1()`, `P2()`, `P3()`: the SU(4) labels.
- `n14()`, `n24()`, `n34()`, `n44()`: the four U(4) labels (only for `u4::irrep`).

Dimension and invariants:

- `dim()`: the irrep dimension (both classes).
- `C2()`, `C3()`, `C4()`: the SU(4) Casimirs of orders 2, 3, 4 (both classes, shift-invariant).
- `C1()`: the first-order Casimir = n_{14}+n_{24}+n_{34}+n_{44} (total quanta), on `u4::irrep` only (SU(4) is U(4) with the U(1) trace generator removed, so `su4::irrep` has no `C1()`).
- `quanta()`: synonym of `C1()`.

Other:

- `zero()`: test for the trivial irrep.
- `conjugate()`: the conjugate irrep (P_3, P_2, P_1).
- `conjugate_phase_power()`: the conjugation phase factor.
- `normalize()`: set $n_{44} = 0$ for `u4::irrep`.
- comparison operators and stream output.
- `writeBinary` / `readBinary`: serialization for binary files.

C. Gelfand states: `u4::gelfand`

(`U4gelfand.h`.) The canonical basis state with U(4) labels $[n_{14}, n_{24}, n_{34}, n_{44}]$ and U(3)⊃U(2)⊃U(1) sublabels $[n_{13}, n_{23}, n_{33}]$, $[n_{12}, n_{22}]$, n_{11}.

Constructor:

- `u4::gelfand(n14,n24,n34,n44,n13,n23,n33,n12,n22,n11)`: build a Gelfand state from its labels.

Label getters and validity check:

- `n14()` ... `n11()`: the ten Gelfand-pattern labels.
- `check()`: test the betweenness (interlacing) condition.
- `normalize()`: shift all quantum numbers by n_{44}.

Subgroup labels:

- `P1()`, `P2()`, `P3()`: the SU(4) labels.
- `Lambdax2()`: 2× the embedded SU(2) angular momentum.
- `MLambdax2()`: 2× its projection.

Conjugation:

- `conjugate()`: the Gelfand state's conjugate.
- `conjugate_phase()`: the accompanying conjugation phase.

Generator matrix elements: Each method returns the matrix element of a U(4) canonical generator E_{ij} acting on a Gelfand state. The suffix identifies the target branch: `_12`/`_22` selects n_{12} or n_{22}, while `_13`/`_23`/`_33` selects n_{13}, n_{23}, or n_{33}. For generators that change both rows, the two suffixes specify the corresponding pair of labels (see Ref. [1]).

- `e11()`, `e22()`, `e33()`, `e44()`: the diagonal weight generators.
- `e12()`: the E_{12} element.
- `e21()`: the E_{21} element.
- `e23_12()`, `e23_22()`: the E_{23} element, per target branch.
- `e32_12()`, `e32_22()`: the E_{32} element, per target branch.
- `e13_12()`, `e13_22()`: the E_{13} element, per target branch.
- `e31_12()`, `e31_22()`: the E_{31} element, per target branch.
- `e34_13()`, `e34_23()`, `e34_33()`: the E_{34} element, per target branch.
- `e43_13()`, `e43_23()`, `e43_33()`: the E_{43} element, per target branch.
- `e24_13_12()`, `e24_13_22()`, `e24_23_12()`, `e24_23_22()`, `e24_33_12()`, `e24_33_22()`: the E_{24} element, per target branch.
- `e42_13_12()`, `e42_13_22()`, `e42_23_12()`, `e42_23_22()`, `e42_33_12()`, `e42_33_22()`: the E_{42} element, per target branch.
- `e14_13_12()`, `e14_13_22()`, `e14_23_12()`, `e14_23_22()`, `e14_33_12()`, `e14_33_22()`: the E_{14} element, per target branch.
- `e41_13_12()`, `e41_13_22()`, `e41_23_12()`, `e41_23_22()`, `e41_33_12()`, `e41_33_22()`: the E_{41} element, per target branch.

The raising-action predicates in `U4basis.h` use the corresponding uppercase names: `E12`, `E23_12`, `E23_22`, `E13_12`, `E13_22`, `E34_13`, `E34_23`, `E34_33`, and the six branches each of `E24` and `E14` with the same paired suffixes listed above. They test whether a candidate final state has the label changes required by that raising-generator branch.

D. Physical states: `u4::physicalvector`

(`U4basis.h`.) The $\mathrm{SU_S}(2) \times \mathrm{SU_T}(2)$ reduced spin-isospin state $|(P_1P_2P_3)\,\zeta\,S\,T\rangle$, where spin and isospin are stored doubled as $SS = 2S$, $TT = 2T$, and ζ is the inner multiplicity.

Constructor:

- `u4::physicalvector(irrep, SS, TT)`: a physical state of doubled spin $SS = 2S$ and isospin $TT = 2T$.

Accessors:

- `irr()`: the underlying irrep.
- `P1()`, `P2()`, `P3()`: the SU(4) labels.
- `n14()` ... `n44()`: the U(4) labels.
- `SS()`, `TT()`: the doubled spin and isospin.
- `zeta()` / `zetamax()`: the inner multiplicity of (S, T).

E. Free functions

Multiplicities:

- `u4::Zeta(irrep,SS,TT)`: inner multiplicity $\zeta_{\max}$ of (S,T).
- `u4::Eta(A,B,C)`: outer multiplicity $\eta_{\max}$ of irrep C in the $A \otimes B$ product (`U4decomposition.h`).

Bases:

- `u4::gelfandBasis(irrep)`: enumerate every Gelfand state of the irrep.
- `u4::physicalBasis(irrep)`: list all physical states with nonzero ζ.
- `u4::findGelfand(g,gArr)`: find the position of a state inside the `gArr` Gelfand basis.

Decomposition:

- `u4::decomposition(A,B)`: the complete Kronecker series, as a (list of C, list of $\eta_{\max}$) pair satisfying $\dim A \dim B = \sum_C \eta_{\max}(C) \dim C$.

Physical generator reduced matrix elements: Each takes two `physicalvector` states (bra, ket) and two `std::vector<int>` out-parameters, and returns an `Eigen::MatrixXd` of shape $\zeta'_{\max} \times \zeta_{\max}$ (`U4generatorsRME.h`).

- `RME_S`, `RME_T`: diagonal in (S,T), with $\langle\|\mathbf{S}\|\rangle = \sqrt{S(S{+}1)(2S{+}1)(2T{+}1)}$ and $\langle\|\mathbf{T}\|\rangle = \sqrt{T(T{+}1)(2T{+}1)(2S{+}1)}$.
- `RME_U`: the rank-$(1,1)$ Gamow–Teller-type tensor ($\Delta S, \Delta T \in \{-1,0,+1\}$).

Transformation: Each routine expands the physical state `p` (at doubled projections `MMS` and `MMT`) over the Gelfand basis `gArr`, returning the canonical-state indices together with an amplitude matrix whose columns are orthonormal, one per ζ.

- `u4::phys_to_cano(p,MMS,MMT,gArr)`: the expansion at a single projection (M_S, M_T).
- `u4::phys_to_cano_allM(p,gArr)`: the expansion over all projections.

F. Coefficient containers

(`Definitions.h`.) Coefficients are grouped into containers whose axes correspond to multiplicity labels. Each container provides coordinate access through `operator()`, flat indexing through `operator[]`, and formatted output through `print()` or `operator<<`:

- `block(i0,...,id)` selects a coefficient by its multiplicity indices, in the order given below. Indices are zero-based and must be smaller than their corresponding `*max()` counts.
- `block[i]` accesses the same data with a single index, $0 \le i <$ `block.size()`. Storage is row-major: the last axis varies fastest. For axis sizes $d_0, \ldots, d_{m-1}$, the flat index is $i = \sum_{j=0}^{m-1} i_j \prod_{k=j+1}^{m-1} d_k$, with an empty product equal to one. For four axes, $i = ((i_0 d_1 + i_1) d_2 + i_2) d_3 + i_3$.
- `data()` returns a const reference to the underlying storage object; `size()` gives the number of coefficients, and `empty()` indicates whether the block contains any coefficient.
- `print(s)` and `s << block` print the multiplicity indices and coefficient values to stream `s`. With no argument, `print()` writes to `std::cout`.

For containers with writable storage, non-const indexing returns a writable reference, while const indexing returns the coefficient by value. The cached `U4U9` database result uses a read-only view; access it through a const container. Storage ownership is discussed at the end of this subsection.

In the notation below, A, B, C, D are U(4) irreps. Labels such as AB, BC, AC, BD, CD refer to selected intermediate irreps, not to the tensor products themselves. We use R for the final irrep in recoupling. The outer multiplicity η distinguishes occurrences of a U(4) irrep in a product; ρ plays the corresponding role for the U(3) subgroup coupling. The inner multiplicity ζ distinguishes repeated spin-isospin pairs within an SU(4) irrep. All multiplicity indices below follow the library's zero-based convention.

`U4U3`: canonical coupling of $A \otimes B \to C$, reduced with respect to U(3): $\langle Aa; Bb \| Cc \rangle_{\rho,\eta}$, where a, b, c are fixed U(3) sublabels $[n_{13}, n_{23}, n_{33}]$. Unlike the U(3) $\supset$ U(2) case, the subgroup coupling itself can have outer multiplicity. The block therefore has two axes, (ρ, η).

- `rhomax()`: multiplicity of the U(3) coupling $a \otimes b \to c$.
- `etamax()`: multiplicity of the U(4) coupling $A \otimes B \to C$.
- `operator()(rho,eta)`: selects a coefficient with flat index $\rho\,\eta_{\max} + \eta$.
- `print(s = std::cout)`: prints the coefficients over all (ρ, η) pairs.

`U4SU2SU2`: physical coupling of $A \otimes B \to C$, reduced with respect to $\mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$: $\langle A\,\zeta_A S_A T_A;\, B\,\zeta_B S_B T_B \| C\,\zeta_C S_C T_C \rangle_\eta$. The spins and isospins are fixed for each block and supplied to the library in doubled form, $SS = 2S$ and $TT = 2T$. The three inner multiplicities and the U(4) outer multiplicity form the four axes.

- `zetaAmax()`: multiplicity of (S_A, T_A) in A.
- `zetaBmax()`: multiplicity of (S_B, T_B) in B.
- `zetaCmax()`: multiplicity of (S_C, T_C) in C.
- `etamax()`: multiplicity of C in $A \otimes B$.
- `operator()(zetaA,zetaB,zetaC,eta)`.
- `print(s = std::cout)`: prints the coefficients over $(\zeta_A, \zeta_B, \zeta_C, \eta)$.

`U4U6`: U-recoupling between $(AB)C$ and $A(BC)$: $U(A, B, R, C;\, AB, \eta_{AB}, \eta_{AB_C};\, BC, \eta_{BC}, \eta_{A_BC})$. The intermediate irreps AB and BC are fixed; the four axes resolve the multiplicities along the two coupling routes.

- `etaABmax()`: multiplicity of $A \otimes B \to AB$.
- `etaAB_Cmax()`: multiplicity of $AB \otimes C \to R$.
- `etaBCmax()`: multiplicity of $B \otimes C \to BC$.
- `etaA_BCmax()`: multiplicity of $A \otimes BC \to R$.
- `operator()(etaAB,etaAB_C,etaBC,etaA_BC)`.
- `print(s = std::cout)`: prints the coefficients over $(\eta_{AB}, \eta_{AB_C}, \eta_{BC}, \eta_{A_BC})$.

`U4Z6`: Z-recoupling between $(AB)C$ and $(AC)B$: $Z(B, A, R, C;\, AB, \eta_{AB}, \eta_{AB_C};\, AC, \eta_{AC}, \eta_{AC_B})$. The intermediate irreps AB and AC are fixed; the four axes resolve the multiplicities along the two coupling routes.

- `etaABmax()`: multiplicity of $A \otimes B \to AB$.
- `etaAB_Cmax()`: multiplicity of $AB \otimes C \to R$.
- `etaACmax()`: multiplicity of $A \otimes C \to AC$.
- `etaAC_Bmax()`: multiplicity of $AC \otimes B \to R$.
- `operator()(etaAB,etaAB_C,etaAC,etaAC_B)`.
- `print(s = std::cout)`: prints the coefficients over $(\eta_{AB}, \eta_{AB_C}, \eta_{AC}, \eta_{AC_B})$.

`U4U9`: 9-U(4) recoupling between $(AB)(CD)$ and $(AC)(BD)$:

$$\begin{bmatrix} A & B & AB & \eta_{AB} \\ C & D & CD & \eta_{CD} \\ AC & BD & R & \eta_{AC_BD} \\ \eta_{AC} & \eta_{BD} & \eta_{AB_CD} & \end{bmatrix}. \tag{1}$$

The last column labels the row couplings, and the last row labels the column couplings. In the code, `etaABCD` denotes η_{AB_CD} for $AB \otimes CD \to R$, whereas `etaACBD` denotes η_{AC_BD} for $AC \otimes BD \to R$. The storage order is $(\eta_{AC}, \eta_{BD}, \eta_{AB_CD}, \eta_{AB}, \eta_{CD}, \eta_{AC_BD})$.

- `etaACmax()`: multiplicity of $A \otimes C \to AC$.

- `etaBDmax()`: multiplicity of $B \otimes D \to BD$.
- `etaABCDmax()`: multiplicity of $AB \otimes CD \to R$.
- `etaABmax()`: multiplicity of $A \otimes B \to AB$.
- `etaCDmax()`: multiplicity of $C \otimes D \to CD$.
- `etaACBDmax()`: multiplicity of $AC \otimes BD \to R$.
- `operator()(etaAC,etaBD,etaABCD,etaAB,etaCD,etaACBD)`.
- `print(s = std::cout)`: prints all six multiplicity indices and the corresponding coefficients.

The classes are templated on their storage type. The first four database containers own their coefficient vectors. With `U4LIB_ENABLE_U4U9_CACHE` on, `get_U4U9` instead returns a container holding a `std::span<const double>` into a thread-private scratch buffer. A later query can overwrite that buffer, so copy the coefficients into owning storage before retaining results across calls; copying the view alone does not retain the data. Otherwise, the returned container owns a `std::vector<double>`.

G. Coefficient databases

Every coupling and recoupling coefficient is evaluated on first use and memoized inside a process-wide singleton database: `u4::wigU4U3db()`, `u4::wigU4SU2SU2db()`, `u4::u4u6db()`, `u4::u4z6db()`, and `u4::u4u9db()`. Each database offers two complementary retrieval styles:

- **Structured container getters** (`get_U4U3`, `get_U4SU2SU2`, `get_U4U6`, `get_U4Z6`, `get_U4U9`): hand back the typed containers described above (`U4U3`, `U4SU2SU2`, `U4U6`, `U4Z6`, `U4U9`), with coordinate access through `operator()`, flat access through `operator[]`, the `*max()` shape queries, and stream printing.
- **Direct buffer getters** (`get`): return the coefficients as a bare buffer, bypassing the container wrapper. The three recoupling databases add a raw-integer overload of `get` that consumes the partition labels directly and derives the intermediate multiplicities on its own.

Taking the databases in turn:

- **Canonical coupling database** (`u4::wigU4U3db()`):
  - `get_U4U3(` $n^a_{14}, n^a_{24}, n^a_{34}, n^a_{44}, n^a_{13}, n^a_{23}, n^a_{33},\ \ldots^b,\ \ldots^c$ `)` $\to$ `U4U3`: the U(4) $\supset$ U(3) reduced coupling indexed by (ρ, η).
  - `get(` $n^a_{14}, n^a_{24}, n^a_{34}, n^a_{44}, n^a_{13}, n^a_{23}, n^a_{33},\ \ldots^b,\ \ldots^c$ `)` $\to$ `Eigen::MatrixXd`: the same block as a $\rho_{\max} \times \eta_{\max}$ matrix; an empty matrix marks a forbidden coupling.
- **Physical coupling database** (`u4::wigU4SU2SU2db()`):
  - `get_U4SU2SU2(` $P^a_1, P^a_2, P^a_3, SS^a, TT^a,\ \ldots^b,\ \ldots^c$ `)` $\to$ `U4SU2SU2`: the physical spin-isospin reduced coupling ($SS = 2S$, $TT = 2T$), indexed by $(\zeta_A, \zeta_B, \zeta_C, \eta)$.
  - `get(` $P^a_1, P^a_2, P^a_3, SS^a, TT^a,\ \ldots^b,\ \ldots^c$ `)` $\to$ `std::vector<double>`: the same block flattened over the $(\zeta_A, \zeta_B, \zeta_C, \eta)$ axes.
- **U-recoupling database** (`u4::u4u6db()`):
  - `get_U4U6(` $A, B, ABC, C, AB,\ \eta_{AB}, \eta_{AB_C},\ BC, \eta_{BC}, \eta_{A_BC}$ `)` $\to$ `U4U6`: recouples $(AB)C \leftrightarrow A(BC)$, indexed by $(\eta_{AB}, \eta_{AB_C}, \eta_{BC}, \eta_{A_BC})$.
  - `get(` $A, B, ABC, C, AB,\ \eta_{AB}, \eta_{AB_C},\ BC, \eta_{BC}, \eta_{A_BC}$ `)` $\to$ `std::vector<double>`: the flat form of length $\eta_{AB}\eta_{AB_C}\eta_{BC}\eta_{A_BC}$.
  - `get(` 24 integers `)` $\to$ `std::vector<double>`: a raw-integer overload taking the partition labels of A, B, ABC, C, AB, BC and resolving the η maxima internally.
- **Z-recoupling database** (`u4::u4z6db()`):

  - `get_U4Z6(` $B, A, ABC, C, AB,\ \eta_{AB}, \eta_{AB_C},\ AC, \eta_{AC}, \eta_{AC_B}$ `)` → `U4Z6`: recouples $(AB)C \leftrightarrow (AC)B$, indexed by $(\eta_{AB}, \eta_{AB_C}, \eta_{AC}, \eta_{AC_B})$ (note the leading B, A exchanges the first pair).
  - `get(` $B, A, ABC, C, AB,\ \eta_{AB}, \eta_{AB_C},\ AC, \eta_{AC}, \eta_{AC_B}$ `)` → `std::vector<double>`: the flat form of length $\eta_{AB}\eta_{AB_C}\eta_{AC}\eta_{AC_B}$.
  - `get(` 24 integers `)` → `std::vector<double>`: a raw-integer overload for the partition labels of B, A, ABC, C, AB, AC and resolving the multiplicities internally.

- **9-**U(4) **recoupling database** (`u4::u4u9db()`):

  - `get_U4U9(` $A, B, C, D, AB, CD, AC, BD, ABCD,\ \eta_{AC}, \eta_{BD}, \eta_{ABCD}, \eta_{AB}, \eta_{CD}, \eta_{ACBD}$ `)` → `U4U9`: the coefficient container; its multiplicity arguments follow the container's axis order, the column channels $(\eta_{AC}, \eta_{BD}, \eta_{ABCD})$ ahead of the row channels $(\eta_{AB}, \eta_{CD}, \eta_{ACBD})$.
  - `get(` $A, B, C, D, AB, CD, AC, BD, ABCD,\ \eta_{AB}, \eta_{CD}, \eta_{ABCD}, \eta_{AC}, \eta_{BD}, \eta_{ACBD}$ `)` → `std::span<const double>` (or `std::vector<double>`): the flat buffer, where the row channels $(\eta_{AB}, \eta_{CD}, \eta_{ABCD})$ instead come first, the reverse of `get_U4U9`, matching the internal stride order.
  - `get(` 36 integers `)` → `std::vector<double>`: a raw-integer overload taking the partition labels of all nine participating irreps, where multiplicities are evaluated internally.

The multiplicity counts passed to the recoupling getters must equal the `Eta` values of the matching channels, or an internal assertion fires. In `get_U4U9` the third count η_{ABCD} belongs to the $(AB)(CD) \to ABCD$ route and the sixth η_{ACBD} to the $(AC)(BD) \to ABCD$ route; interchanging them raises the same assertion. A forbidden or vanishing coupling returns an empty container or buffer, which may be queried for its shape but not indexed. Each database exposes a `clear()` method—and `u4::clear()` empties all five at once—that drops the cached blocks while keeping the allocated thread workspaces.

H. Conventions and normalization

Phase and normalization conventions follow Ref. [1]; documented values and the generation procedure are in `test/reference/`.

- **Doubled spin-isospin:** All spin and isospin labels are doubled: $SS = 2S$, $TT = 2T$ (and $MMS = 2M_S$, $MMT = 2M_T$).

- **Multiplicity indexing:** The `Zeta` and `Eta` routines report the counts, and every container accessor is indexed from zero up to the corresponding maximum $(0, \ldots, \max{-}1)$.

- **Canonical-coupling phase:** Within each η block the highest-weight reduced coefficients set both phase and normalization. When $\eta_{\max} > 1$ the solver returns one admissible orthonormal null-space basis, and all coefficients carrying η must use the same basis choice. Under conjugation a Gelfand state obeys $C\,|g\rangle = (-)^{n_{14}+n_{34}-n_{13}-n_{23}-n_{33}-n_{12}-n_{22}-n_{11}}\,|g.\text{conjugate}()\rangle$ (`gelfand::conjugate_phase()`).

- **Physical-to-canonical transformation:** A physical state $|(P_1P_2P_3)\,\zeta\,S\,M_S\,T\,M_T\rangle$ is expanded in canonical Gelfand states. The highest-weight state $|(P_1P_2P_3)\,\zeta\,S\,S\,T\,T\rangle$ is the null space of S_+ and T_+ [1]; S_- and T_- fix the lower-M phases within a ζ multiplet. For $P_1 \neq P_3$ the conjugation preserves ζ,

$$C\,|(P_1P_2P_3)\,\zeta\,S\,M_S\,T\,M_T\rangle = \epsilon\,(-)^{S-M_S+T-M_T}\ |(P_3P_2P_1)\,\zeta\,S\,{-}M_S\,T\,{-}M_T\rangle\,, \quad \epsilon = \begin{cases} (-)^{P_1+P_2+P_3}, & P_1 > P_3, \\ (-)^{P_2}, & P_1 < P_3. \end{cases}$$

  For a self-conjugate irrep ($P_1 = P_3$), conjugation instead acts as a projection-independent symmetric matrix D ($D^2 = \mathbb{1}$) that may mix ζ. The direction-dependent ϵ reflects $C_{(P_3P_2P_1)}C_{(P_1P_2P_3)} = (-)^{P_1+P_3}\mathbb{1}$, which is $+1$ whenever $P_1 + P_3$ is even and -1 otherwise; one may find an analogy in the SU(2) conjugation factor $C_jC_j = (-)^{2j}$.

- **Generator reduced matrix element factor:** `RME_S`/`RME_T`/`RME_U` carry an extra $\sqrt{(2S'{+}1)(2T'{+}1)}$ factor.

- **Assertion checks.** Invalid labels or out-of-range indices abort via assertion.

IV. PYTHON BINDINGS

The `u4lib` extension module reproduces the C++ API one-to-one. Build it with `make python` using the interpreter whose ABI matches the extension, and keep `u3lib`'s `python/` on the import path, since `u4lib` depends on it. The irreps (`su4.irrep`, `irrep`), `gelfand`, `physicalvector`, the free functions `Eta`, `Zeta`, `decomposition`, `gelfandBasis`, `physicalBasis`, the generator reduced matrix elements `RME_S`/`RME_T`/`RME_U`, the transformations `phys_to_cano`/`phys_to_cano_allM`, and the five database accessors all carry the names and semantics of Sec. III. Each structured container supports `__call__` for multiplicity-indexed access, exposes its flat `data()` as a NumPy array, and prints itself; querying a database before `initialize()` or after `finalize()` raises `RuntimeError`. The `U4U9` container, only a scratch-buffer view in C++, is copied automatically so that the Python object owns its data. The examples in the Jupyter notebook `python/tutorial.ipynb` cover irreps, bases, coupling, and recoupling. Consult `python/BUILD_NOTES.md` for ABI and toolchain requirements.

V. AI-READY INTERFACE

A. Skill catalog and orchestration

The `agents/` directory holds a catalog of skills that steer an AI assistant through `u4lib` calculations. An index (`agents/AGENTS.md`) points to shared runtime instructions and one skill per capability, each documenting the inputs, the exact library call, the relevant conventions, and a numerical check. An assistant with code-execution access follows these to drive the locally built library, and no network service is involved.

Setup is handled by the `install` skill (`agents/install/SKILL.md`). It first checks whether the library is already built, and otherwise runs `make config`—which resolves every dependency by inheriting `u3lib`'s toolchain and needs no manual environment variables—followed by `make` and `make test` (a clean run prints `PASS: all test executables`); `make python` adds the Python extension when required. Once the build passes, the calculation skills become usable.

For a calculation, an AI assistant maps the request onto a skill, prepares the inputs in the library's conventions, invokes the C++ or Python routine, and returns the results. The numerical work stays inside the library—the assistant neither derives nor approximates coefficients, Casimirs, or matrix elements—and when a request is ambiguous or asks for an undefined quantity it seeks clarification or explains the problem, e.g., the first-order Casimir of an SU(4) irrep, which is not defined.

Table II pairs each skill with a representative prompt whose parenthetical answer was verified against the library. Spin and isospin inputs use the doubled labels $SS = 2S$ and $TT = 2T$.

B. Benchmark objectives

The benchmark tests whether the catalog helps an agent turn a U(4) calculation request into a working program. Correctness is assessed separately from the resources spent reaching a solution. We evaluate two hypotheses:

1. Resource efficiency: Ready access to argument conventions and usage examples should reduce source-code searches and unsuccessful implementation attempts, lowering token use, runtime, and API cost.

2. Dependence on task complexity: Guidance about memory ownership and thread safety should become more useful when the task moves beyond Python bindings to C++ storage management and parallel execution.

C. Experimental strategy

1. *Benchmark Tasks*

The three tasks exercise different parts of the library's interface and place progressively greater demands on storage and concurrency:

- Task 1: Python Racah Recoupling Engine (`consumer/recouple.py`). Write a Python utility using the pybind11 bindings to evaluate 6-U(4) recoupling coefficients via `u4lib.u6recoupling` and `u4lib.z6recoupling`. The implementation must check the unitary symmetry relations among fundamental representations (for instance, $(1,0,0)\otimes(1,0,0)\otimes(1,0,0)\to(3,0,0)$) and export structured numerical outputs. *Architectural complexity: Low* (declarative Python bindings, automatic garbage collection, single-threaded execution).

Skill	Example prompt (expected result)
`dim`	"Dimension of the SU(4) irrep $(1,0,1)$ – equivalently the U(4) irrep $[2,1,1,0]$?" (15, the adjoint; shift-invariant)
`casimirs`	"C_2, C_3, C_4 for the SU(4) adjoint $(1,0,1)$, and C_1, C_2, C_3, C_4 for the U(4) irrep $[2,1,1,0]$." (SU(4): 16, 0, 72; U(4): C_1=4, 16, 0, 72 – C_1 is U(4)-only)
`cano-list`	"Enumerate the canonical U(4) Gelfand basis of $(1,0,0)$ with each state's weights and subgroup labels." (4 states = dim)
`phys-list`	"List the physical spin-isospin basis (ζ, S, T) of $(2,0,0)$ and verify the dimension sum rule." ((S,T)=(1, 1), (0, 0), each ζ=1; $\sum \zeta(2S{+}1)(2T{+}1)$=10)
`cano-generators`	"Act with E_{12} on the Gelfand state $\lvert 2,0,0,0;2,0,0;2,0;1\rangle$ of $(2,0,0)$; give the resulting state and matrix element." ($\lvert 2,0,0,0;2,0,0;2,0;2\rangle$ with $\sqrt{2}$)
`phys-generators`	"Compute $\langle(2,0,0)\,\zeta{=}1\,S{=}1\,T{=}1\|\mathbf{S}\|(2,0,0)\,\zeta{=}1\,S{=}1\,T{=}1\rangle$ and $\langle\|\mathbf{U}\|\rangle$ on the same state." ($\sqrt{18} \approx 4.243$ and 3)
`phys-to-cano`	"Express $\lvert(2,0,0)\,\zeta{=}1\,S{=}1\,M_S{=}1\,T{=}1\,M_T{=}1\rangle$ in the canonical Gelfand basis." (single state $\lvert 2,0,0,0;2,0,0;2,0;2\rangle$, amplitude 1)
`inner-mult`	"Spin-isospin multiplicity ζ of (S,T)=(1, 1) in the adjoint $(1,0,1)$?" (1)
`outer-mult`	"Outer multiplicity η of $(1,0,1)$ in $(1,0,1) \otimes (1,0,1)$?" (2)
`decomposition`	"Full decomposition of $(1,0,0) \otimes (1,0,0)$ with outer multiplicities; check the dimension sum rule." ($(2,0,0){+}(0,1,0)$, $\sum \eta \cdot \dim = 16$)
`cano-coupling`	"SU(4)$\supset$U(3) reduced coupling for highest-weight states $[1,0,0,0]\,[1,0,0]\times[1,0,0,0]\,[1,0,0] \to [2,0,0,0]\,[2,0,0]$ of $(1,0,0)\times(1,0,0)\to(2,0,0)$; verify it equals 1."
`phys-coupling`	"Spin-isospin coupling $(1,0,0)\,\zeta{=}1\,S{=}\frac{1}{2}\,T{=}\frac{1}{2} \times (1,0,0)\,\zeta{=}1\,S{=}\frac{1}{2}\,T{=}\frac{1}{2} \to (2,0,0)\,\zeta{=}1\,S{=}1\,T{=}1$; verify it equals 1."
`u6recoupling`	"U-recoupling for $(1,0,0)\otimes(1,0,0)\otimes(1,0,0)\to(3,0,0)$ through intermediate irreps $AB = (2,0,0)$ and $BC = (2,0,0)$." (1)
`z6recoupling`	"Z-recoupling between $(AB)C$ and $(AC)B$ routes for $(1,0,0)\otimes(1,0,0)\otimes(1,0,0)\to(3,0,0)$ with intermediate irreps $AB = (2,0,0)$ and $AC = (2,0,0)$." (1)
`u9recoupling`	"9-coefficient recoupling for four fundamentals $(1,0,0)$ coupling to $(4,0,0)$ with intermediate irreps $AB = CD = AC = BD = (2,0,0)$." (1)
`conjugation`	"Conjugate the SU(4) irrep $(2,0,0)$ and a Gelfand state of it, reporting the phases." (irrep $\to (0,0,2)$, phase power 3)

TABLE II: The `u4lib` agent skills with a representative prompt for each.

- Task 2: C++ 9-U(4) Batch Exporter (`consumer/u9_batch.cpp`). Build a C++ batch exporter for 9-U(4) recoupling blocks via `u4::u4u9db()`. The program must parse integer input specifications for nine participating irreps, compute intermediate outer multiplicities η, retain coefficient buffers across requests without aliasing, configure build dependencies, and serialize block and entry records. *Architectural complexity: Medium* (C++ typing, header inclusion, manual workspace lifecycle management, and partition label handling).
- Task 3: C++ OpenMP Multi-threaded Concurrency (`consumer/coupling_batch.cpp`). Implement parallel canonical and physical coupling evaluations with OpenMP directives (`#pragma omp parallel for`). The implementation must give each thread its own database cache and scratch state through per-thread initialization, guard shared output, set the required compiler flags (`-fopenmp`), and scale across cores. *Architectural complexity: High* (shared-memory concurrency, thread-private state isolation, race-hazard avoidance).

2. Isolated workspace preparation

Runs started in independent temporary directories, each with its own consumer build environment. Files and build products from earlier trials were not reused. The workspace snapshot determined what the agent could consult:

- Control (standard prompts, zero skills): Source files, C++ headers, and build scripts were available, but the `agents/` directory was removed.
- Treatment (domain skills): The workspace also contained the repository's skill catalog (`agents/skills/`), indexed by `agents/AGENTS.md`. The prompt instructed the agent to consult it for conventions and usage patterns.

TABLE III: Performance of Control (standard prompts, zero skills) versus Treatment (repository-local domain skills) on `u4lib` with Claude Sonnet 5.5, over $N = 10$ matched pairs per task (60 runs in total). Token counts, runtimes, and costs are given as Mean ± Standard Deviation, and significance follows from two-tailed Welch's t-tests.

Task	Condition	Pass Rate	Total Tokens	Time (s)	Cost ($)	Token Δ	Time Δ
Task 1: Python	Control (N=10)	90.0% (9/10)	$650,386 \pm 304,782$	97.35 ± 42.34	$0.3949	—	—
	Treatment (N=10)	100% (10/10)	$318,884 \pm 109,257$	62.04 ± 42.67	$0.2377	**−51.0%****	**−36.3%**
Task 2: C++ Batch	Control (N=10)	90.0% (9/10)	$693,475 \pm 230,556$	117.69 ± 24.53	$0.4058	—	—
	Treatment (N=10)	100% (10/10)	$311,188 \pm 75,098$	77.24 ± 25.88	$0.2446	**−55.1%*****	**−34.4%****
Task 3: C++ OpenMP	Control (N=10)	80.0% (8/10)	$807,737 \pm 231,065$	157.59 ± 69.85	$0.4628	—	—
	Treatment (N=10)	100% (10/10)	$291,843 \pm 69,332$	70.20 ± 12.99	$0.2443	**−63.9%*****	**−55.4%****
Grand Aggregate	Control (N=30)	86.7% (26/30)	$717,199 \pm 257,789$	124.21 ± 53.90	$0.4212	—	—
	Treatment (N=30)	100% (30/30)	$307,305 \pm 84,146$	69.83 ± 29.41	$0.2422	**−57.1%*****	**−43.8%*****

$^{**}p < 0.01$ (Task 1: $t_{\text{tok}} = -3.24$, p=0.0077; Task 2 time: $t = -3.59$, p=0.0021; Task 3 time: $t = -3.89$, p=0.0032);

$^{***}p < 0.001$ (Task 2 tok: $t = -4.99$, $p = 4.2 \times 10^{-4}$; Task 3 tok: $t = -6.76$, $p = 3.7 \times 10^{-5}$);

Grand Aggregate: $t_{\text{tok}} = -8.28$, $t_{\text{time}} = -4.85$; Welch's two-tailed t-test.

3. *Completion and validation*

Agents could read and edit files, inspect directories, and run shell commands without intervention. A run ended when the agent declared completion or used 50 steps. The workspace was then frozen for evaluation against the hidden tests. The grader compared numerical output with exact coupling and recoupling values for several irreps, allowing a tolerance of 10^{-12}. We checked the grader itself including phase inversions, memory leaks, uninitialized reads, and race conditions. All such validation cases were classified correctly: sensitivity and specificity were 100%, with no false positives or negatives.

4. *Model and Statistical Protocol*

Every trial used Anthropic's Claude Sonnet 5.5 (`claude-sonnet-5-5`) in the same autonomous test environment. Each of the three tasks contributed $N = 10$ matched Control/Treatment pairs, giving 60 runs overall. For every run, four primary metrics were recorded:

1. Functional pass rate (%): Percentage of runs passing all hidden unit tests.
2. Total token consumption: Cumulative sum of prompt tokens, cache-read tokens, and completion tokens consumed across all agent steps.
3. Wall-clock latency (s): Total elapsed real time from task dispatch to final agent termination, measured externally by the evaluation harness.
4. Monetary cost (USD): Total API billing incurred, computed from official token pricing.

Differences were assessed using two-tailed Welch's t-tests (unequal variance assumption), reporting t-statistics, degrees of freedom, and p-values.

D. Results and discussion

1. *Accuracy and resource use*

All Treatment programs passed the hidden tests; Control programs failed in four runs. Resource use was also lower with the skill catalog. The totals for the 30 matched pairs are reported below, with results in Table III and Fig. 1:

- Token Consumption: with the skill catalog, mean token consumption dropped from $717,199 \pm 257,789$ to $307,305 \pm 84,146$ tokens, a 57.1% aggregate reduction ($t = -8.28$, $p = 9.05 \times 10^{-10}$).

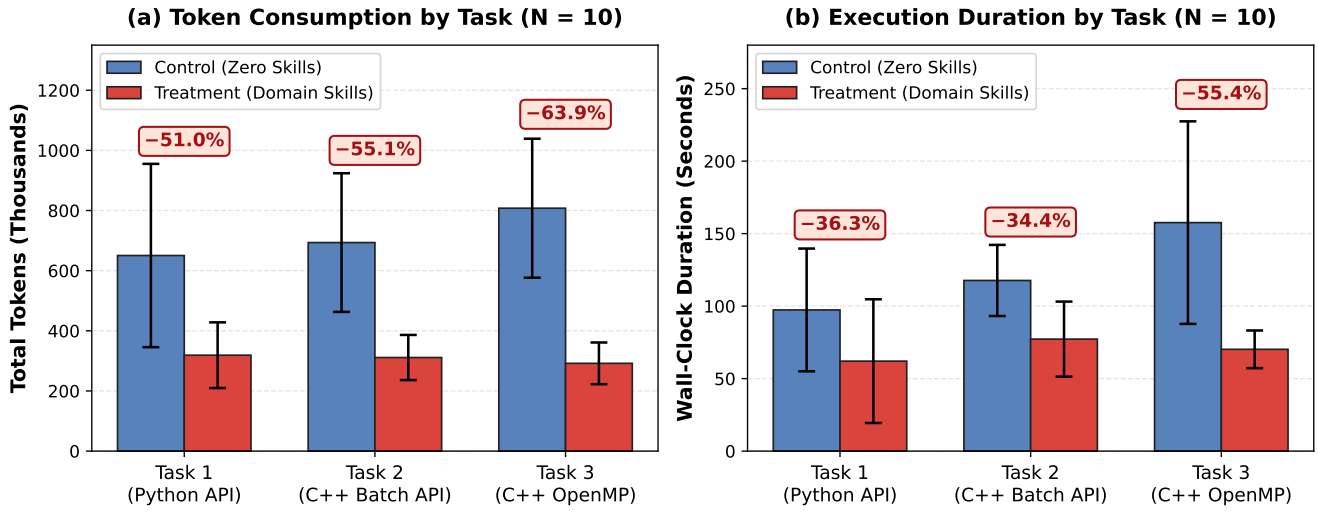


FIG. 1: Autonomous-agent performance under Control (standard prompts, zero skills) and Treatment (repository-local domain skills) on `u4lib`, over $N = 10$ matched pairs per task (60 runs total). (a) Per-task token consumption (thousands), with reductions of 51.0%, 55.1%, and 63.9%. (b) Per-task wall-clock runtime (seconds), with speedups of 36.3%, 34.4%, and 55.4%. Error bars show ±1 sample standard deviation.

- Execution Latency: mean wall-clock time shortened from 124.21 ± 53.90 s to 69.83 ± 29.41 s, a 43.8% aggregate speedup ($t = -4.85$, $p = 1.52 \times 10^{-5}$).
- API Cost: average per-task inference cost went from \$0.4212 to \$0.2422 (−42.5%, $t = -9.52$, $p = 2.49 \times 10^{-11}$).
- Reliability: all 30 Treatment runs passed (10/10 on each of the three tasks), whereas Control runs failed four times across the suite (9/10, 9/10, and 8/10 on Tasks 1–3), an aggregate pass rate of 86.7%.

2. Dependence on task complexity

Across the three tasks, the relative token savings grow with the intended rise in complexity, while the execution-time savings peak on the parallel C++ task without following a strictly monotonic trend:

$$\Delta_{\text{tokens}} = -51.0\% \text{ (Task 1)} \longrightarrow -55.1\% \text{ (Task 2)} \longrightarrow -63.9\% \text{ (Task 3)},$$
$$\Delta_{\text{time}} = -36.3\% \text{ (Task 1)} \longrightarrow -34.4\% \text{ (Task 2)} \longrightarrow -55.4\% \text{ (Task 3)}.$$

Token savings increased from the Python task to the parallel C++ task. Runtime savings did not increase at every step, although Task 3 had the largest reduction in both measures. This is consistent with a benefit from documenting implementation constraints. The run logs help identify sources of challenges the agents faced:

1. Task 1 (argument and channel order): Control agents searched headers and tried different signatures to determine how the Python interface represents U(4) labels and distinguishes U- from Z-recoupling. In one Control run, the agent inverted intermediate coupling channels, producing sign errors that failed validation. Treatment agents used the skill catalog's argument order and usage patterns, completing the task in an average of 318,884 tokens (a 51.0% reduction, $p = 0.0077$) with zero failures.

2. Task 2 (representation labels and retained buffers): Task 2 requires evaluating recoupling coefficients via `u4::u4u9db()` and retaining all blocks in memory before serialization. Control agents had difficulty distinguishing SU(4) labels ($P_1P_2P_3$) and U(4) partition labels ($n_{14}, n_{24}, n_{34}, n_{44}$). In the second Control run, the unguided agent mistakenly passed SU(4) labels into the `u4::irrep` constructor with $n_{44} = 0$, violating the betweenness condition and triggering a fatal assertion failure. The logs also show repeated scratch-buffer aliasing errors. Meanwhile, Treatment agents used the catalog's constructor examples and storage instructions, with a 55.1% token reduction ($p = 4.2 \times 10^{-4}$) and a 100% pass rate.

3. Task 3 (shared database state): Parallel U(4) coupling requires care because the coefficient databases keep per-thread cache and scratch state. Control implementations must share a single database across threads in

`#pragma omp parallel for` blocks; hence, not initializing each thread's state or guarding shared output produced race conditions, memory corruption, and non-deterministic crashes, causing two runs to fail. Even successful Control runs required repeated debugging (averaging 157.59 s and 807, 737 tokens). Treatment implementations followed the catalog's OpenMP rules—initializing each thread's state and guarding shared I/O. They produced deterministic output in 70.20 s and 291, 843 tokens, representing a 63.9% token reduction ($p = 3.7 \times 10^{-5}$) and 55.4% speedup ($p = 0.0032$).

VI. BUILD AND INSTALLATION

One must install `u3lib` before configuring `u4lib`. The U(4) headers in `include/` can then be included directly in user programs. Source code is available at https://github.com/Deep-SANE/u4lib.

Run `make config` or `./config/configure.sh` to locate `u3lib` and resolve compiler flags and dependency paths. Configuration is written to `config/config.generated.mk`; no manual environment variables are required. Use `make` to compile the example programs to `bin/`, `make test` to rebuild and execute the regression suite, and `make python` for the pybind11 extension `python/u4lib.so`. The tests cover irrep dimensions, Gelfand structure, $\mathrm{SU_S}(2) \times \mathrm{SU_T}(2)$ branching, coupling orthonormality in ζ and η, conjugation identities, and reference values. Numerical results are compared within tolerances. All core external dependencies are provided by `u3lib`, see Ref. cite u3lib paper here .

Acknowledgments

This work is dedicated to the memory of J. P. Draayer, who passed away on September 13, 2026 and whose pioneering work in algebraic computational nuclear physics—from Elliott's SU(3) model and the Symplectic Sp(3, R) model to the *ab initio* Symmetry-Adapted No-Core Shell Model and, most recently, Wigner's SU(4) supermultiplet model—inspired this framework. The authors also gratefully acknowledge F. Pan and D. Kekejian for valuable discussions. PD was supported by the Institute of International Education through the Quad Fellowship, Louisiana State University, and Quantum CodeX. DL and TD were funded by Quantum CodeX.

[1] P. Dang, J. P. Draayer, F. Pan, T. Dytrych, D. Langr, *et al.*, "Coupling and Recoupling Coefficients for Wigner's U(4) Supermultiplet Symmetry," Eur. Phys. J. Plus **139**, 933 (2024), doi:10.1140/epjp/s13360-024-05581-6.

[2] E. P. Wigner, "On the Consequences of the Symmetry of the Nuclear Hamiltonian on the Spectroscopy of Nuclei," Phys. Rev. **51**, 106 (1937), doi:10.1103/PhysRev.51.106.

[3] D. B. Kaplan and M. J. Savage, "The spin-flavor dependence of nuclear forces from large-N QCD," Phys. Lett. B **365**, 244 (1996), doi:10.1016/0370-2693(95)01277-X.

[4] D. Lee *et al.*, "Hidden Spin-Isospin Exchange Symmetry," Phys. Rev. Lett. **127**, 062501 (2021), doi:10.1103/PhysRevLett.127.062501.

[5] V. S. Timóteo, S. Szpigel and E. Ruiz Arriola, "Symmetries of the similarity renormalization group for nuclear forces," Phys. Rev. C **86**, 034002 (2012), doi:10.1103/PhysRevC.86.034002.

[6] M. Cauvin, V. Gillet, F. Soulmagnon and M. Danos, "Mass formula based on SU(4)," Nucl. Phys. A **361**, 192 (1981), doi:10.1016/0375-9474(81)90476-0.

[7] P. Van Isacker, O. Juillet and B. K. Gjelsten, "A nuclear mass formula based on SU(4) symmetry," Foundations of Physics **27**, 1047 (1997), doi:10.1007/BF02551152.

[8] P. Dang, E. Espinoza, X. Wan, M. Negro et al., "Bridging Ab Initio Symmetries and Global Nuclear Masses with Interpretable Neural Networks," arXiv:2606.28287.

[9] E. P. Wigner, "On Coupling Conditions in Light Nuclei and the Lifetimes of β-Radioactivities," Phys. Rev. **56**, 519 (1939), doi:10.1103/PhysRev.56.519.

[10] J.-I. Fujita and K. Ikeda, "Existence of Isobaric States and Beta Decay of Heavier Nuclei," Nucl. Phys. **67**, 145 (1965), doi:10.1016/0029-5582(65)90119-7.

[11] S. S. Li Muli, T. R. Djärv, C. Forssén and D. R. Phillips, "Role of Spin-Isospin Symmetries in Nuclear β-Decays," Phys. Rev. Lett. **136**, 242501 (2026), doi:10.1103/vk51-b476.

[12] Yu. S. Lutostansky and V. N. Tikhonov, "Charge-exchange resonances and restoration of the Wigner SU(4)-symmetry in heavy and superheavy nuclei," EPJ Web Conf. **107**, 06004 (2016), doi:10.1051/epjconf/201610706004.

[13] D. Lee, "Lattice Effective Field Theory Simulations of Nuclei," Ann. Rev. Nucl. Part. Sci. **75**, 109 (2025), doi:10.1146/annurev-nucl-101918-023343.

[14] P. Van Isacker, O. Juillet, and F. Nowacki, "Pseudo-SU(4) Symmetry in pf-Shell Nuclei," Phys. Rev. Lett. **82**, 2060 (1999), doi:10.1103/PhysRevLett.82.2060.

[15] V. K. B. Kota and R. Sahu, "Proxy-SU(4) symmetry in A = 60–90 region," Phys. Scr. **99**, 065306 (2024), doi:10.1088/1402-4896/ad46ca.

[16] J. P. Elliott, "Collective motion in the nuclear shell model. I. Classification schemes for states of mixed configurations," Proc. R. Soc. A **245**, 128 (1958), doi:10.1098/rspa.1958.0072.

[17] J. P. Elliott, "Collective Motion in the Nuclear Shell Model. II. The Introduction of Intrinsic Wave-Functions," Proc. R. Soc. A **245**, 562 (1958), doi:10.1098/rspa.1958.0101.

[18] T. Dytrych *et al.*, "Collective Modes in Light Nuclei from First Principles," Phys. Rev. Lett. **111**, 252501 (2013), doi:10.1103/PhysRevLett.111.252501.

[19] T. Dytrych *et al.*, "Physics of Nuclei: Key Role of an Emergent Symmetry," Phys. Rev. Lett. **124**, 042501 (2020), doi:10.1103/PhysRevLett.124.042501.

[20] A. E. McCoy, M. A. Caprio, T. Dytrych, and P. J. Fasano, "Emergent $\mathrm{Sp}(3,\mathbb{R})$ Dynamical Symmetry in the Nuclear Many-Body System from an Ab Initio Description," Phys. Rev. Lett. **125**, 102505 (2020), doi:10.1103/PhysRevLett.125.102505.

[21] K. T. Hecht and S. P. Pang, "On the Wigner Supermultiplet Scheme," J. Math. Phys. **10**, 1571 (1969), doi:10.1063/1.1665007.

[22] J. P. Draayer, "$\mathrm{U}(4) \supset \mathrm{SU}(2) \otimes \mathrm{SU}(2)$ Projection Techniques," J. Math. Phys. **11**, 3225 (1970), doi:10.1063/1.1665119.

[23] A. Partensky and C. Maguin, "The $\mathrm{SU}(4) \supset \mathrm{SU}(2) \otimes \mathrm{SU}(2)$ chain," J. Math. Phys. **19**, 511 (1978), doi:10.1063/1.523687.

[24] D. J. Rowe and J. Repka, "The Representations and Coupling Coefficients of su(n); Application to su(4)," Found. Phys. **27**, 1179 (1997), doi:10.1007/bf02551440.

[25] S. Quintero, R. Henao, and J. P. Valencia, "Automated Calculation of Spin–Isospin Branching Rules for SU(4) Irreps," Comput. Phys. Commun. **323**, 110118 (2026), doi:10.1016/j.cpc.2026.110118.

[26] F. Pan, Y. Wu, A. Li, Y. Zhang, L. Dai, *et al.*, "An algebraic projection procedure for construction of the basis vectors of irreducible representations of U(4) in the $\mathrm{SU_S}(2)\otimes\mathrm{SU_T}(2)$ basis," Eur. Phys. J. Plus **138**, 662 (2023), doi:10.1140/epjp/s13360-023-04261-1.

[27] F. Pan, L. Dai, and J. P. Draayer, "Wigner coefficients of $\mathrm{U}(4) \supset \mathrm{SU_S}(2) \otimes \mathrm{SU_T}(2)$," Nucl. Phys. A **1040**, 122746 (2023), doi:10.1016/j.nuclphysa.2023.122746.

[28] F. Pan, L. Dai, and J. P. Draayer, "Elementary coupling coefficients for the Wigner supermultiplet symmetry," Comput. Phys. Commun. **296**, 109025 (2024), doi:10.1016/j.cpc.2023.109025.

[29] M. Kuhn and H. Walliser, "Program for calculating SU(4) Clebsch–Gordan coefficients," Comput. Phys. Commun. **179**, 733 (2008), doi:10.1016/j.cpc.2008.06.009.

[30] A. Alex, M. Kalus, A. Huckleberry, and J. von Delft, "A numerical algorithm for the explicit calculation of SU(N) and $\mathrm{SL}(\mathrm{N},\mathbb{C})$ Clebsch–Gordan coefficients," J. Math. Phys. **52**, 023507 (2011), doi:10.1063/1.3521562.

[31] P. Dang, D. Langr, T. Dytrych, J.P. Draayer, D. Kekejian, "Unmasking Hidden Wigner's Symmetry from First Principles," arXiv:2604.27144.

[32] A. Krastenov, "Sparse Null Space Solver in C++," Master's thesis, Faculty of Information Technology, Czech Technical University in Prague (2026).

[33] J. P. Draayer and Y. Akiyama, "Wigner and Racah coefficients for SU_3," J. Math. Phys. **14**, 1904 (1973), doi:10.1063/1.1666267.

[34] Y. Akiyama and J. P. Draayer, "A User's Guide to Fortran Programs for Wigner and Racah Coefficients of SU_3," Comput. Phys. Commun. **5**, 405 (1973), doi:10.1016/0010-4655(73)90077-5.

[35] P. Dang, J. P. Draayer, F. Pan, and K. S. Becker, "New Procedure for Evaluation of U(3) Coupling and Recoupling Coefficients," Eur. Phys. J. Plus **139**, 704 (2024), doi:10.1140/epjp/s13360-024-05501-8.

[36] J. Zhang, J. Vala, S. Sastry, and K. B. Whaley, "Geometric theory of nonlocal two-qubit operations," Phys. Rev. A **67**, 042313 (2003), doi:10.1103/PhysRevA.67.042313.

[37] A. V. Gorshkov, M. Hermele, V. Gurarie, C. Xu, P. S. Julienne, *et al.*, "Two-orbital SU(N) magnetism with ultracold alkaline-earth atoms," Nat. Phys. **6**, 289 (2010), doi:10.1038/nphys1535.

[38] A. F. Young *et al.*, "Spin and valley quantum Hall ferromagnetism in graphene," Nat. Phys. **8**, 550 (2012), doi:10.1038/nphys2307.