IntegrateUnitary.jl: A `Julia` package for symbolic integration over Haar measures

ŁUKASZ PAWELA, Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Poland
ZBIGNIEW PUCHAŁA, Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Poland

Symbolic integration over the Haar measure of compact groups is a computational cornerstone in quantum information science and random matrix theory. We present `IntegrateUnitary.jl`, a comprehensive Julia package for computing exact expectations of polynomial functions over a wide range of compact groups ($U(d)$, $O(d)$, $Sp(d)$, and $SU(d)$ for balanced polynomials), circular and Gaussian ensembles, Ginibre ensembles, permutation groups, random pure states, and unitary t-designs. The package provides a fully open-source implementation of the Weingarten calculus and Wick contractions with broad symbolic-d support for entry-wise and trace-polynomial integrals, while selected workflows currently require concrete integer dimensions (including higher pure trace moments $|\mathrm{tr}(U)|^{2k}$ for $k > 1$ and HCIZ with `SymbolicMatrix` inputs, and direct matrix-valued integration of `SymbolicMatrix`/`SymbolicMatrixProduct` expressions), automatic asymptotic expansions, a high-level symbolic trace interface that reconstructs Weingarten graphs from index-free expressions, and a bridge to `ITensors.jl` for tensor network averaging. We discuss the underlying algorithms, including the Murnaghan-Nakayama rule and symplectic-orthogonal duality, and demonstrate that the package efficiently handles high-degree moments and quantum information metrics.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; • **Mathematics of computing → Mathematical software**.

Additional Key Words and Phrases: Symbolic integration, Haar measure, Weingarten calculus, Julia programming language, quantum information.

ACM Reference Format:
Łukasz Pawela and Zbigniew Puchała. 2026. IntegrateUnitary.jl: A `Julia` package for symbolic integration over Haar measures. *ACM Trans. Math. Softw.* 1, 1 (June 2026), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Integration over the Haar measure of compact groups is a cornerstone technique in mathematical physics, with profound implications in random matrix theory (RMT) [Forrester 2010; Livan et al. 2018; Mehta 2004] and quantum information theory (QIT) [Nielsen and Chuang 2010; Watrous 2018]. The ability to compute moments of random matrices enables the exact characterization of "typical" properties of quantum systems, such as the average entanglement entropy of subsystems, a concept famously pioneered by Page [Page 1993]. Furthermore, these integrals are essential for studying quantum chaos, scrambling, and the design of quantum circuits that approximate randomness, known as unitary t-designs [Brandao et al. 2016].

Historically, the evaluation of these integrals was a formidable challenge. While the defining properties of Haar measure were established by Hurwitz and Weyl in the early 20$^{\text{th}}$ century, a systematic method for evaluating polynomial integrals emerged later with the work of Weingarten [Weingarten 1978]. He showed that asymptotic integrals could be expressed as sums over permutations involving a weight function now called the Weingarten function. This theory was rigorously detailed and extended to finite d by Collins [Collins 2003]. The seminal work of Collins and Śniady [Collins and

Authors' Contact Information: Łukasz Pawela, lpawela@iitis.pl, Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland; Zbigniew Puchała, z.puchala@iitis.pl, Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland.

2026. Manuscript submitted to ACM

Śniady 2006] unified the treatment for the unitary, orthogonal, and symplectic groups, providing explicit combinatorial formulas for the Weingarten functions in terms of characters of the symmetric group (or related structures).

Despite the existence of these closed-form solutions, their practical evaluation remains non-trivial due to the combinatorial complexity of the symmetric group sums and the intricacies of representation theory. Several software packages have been developed to automate these calculations. Notable examples include `RTNI` [Fukuda et al. 2019], a Mathematica package that offers robust functionality but relies on the proprietary Wolfram engine, and `Haarpy` [Cardin et al. 2024], a Python library that provides accessible symbolic integration. Earlier work by Puchała and Miszczak [Puchała and Miszczak 2017] also explored symbolic integration algorithms. A detailed feature comparison is provided in Table 1, and a direct performance comparison with `Haarpy` and `RTNI` is presented in Sections 6.4 and 6.5.

However, existing tools often face limitations in extensibility, performance, or the ability to handle symbolic dimensions d natively across a broad class of workflows. Furthermore, the translation from high-level, coordinate-free mathematical expressions (e.g., traces of products) to the index-based tensor contractions required by Weingarten calculus is often left to the user, leading to error-prone manual preprocessing. `IntegrateUnitary.jl` addresses these issues and focuses on maximizing performance and symbolic flexibility within the compact group integration domain. A primary distinction lies in `IntegrateUnitary.jl`'s deep integration with the Julia `Symbolics.jl` ecosystem, treating dimensions d as native symbolic variables rather than requiring separate symbolic algebra backends or manual substitutions. This diverse ensemble support, combined with specialized utilities for partial traces of symbolic subsystems, makes `IntegrateUnitary.jl` a versatile environment for both theoretical derivations and complex numerical experiments.

`IntegrateUnitary.jl` offers a pure Julia implementation of the Weingarten calculus for the unitary $U(d)$, orthogonal $O(d)$, and symplectic $Sp(d)$ groups, as well as $SU(d)$ for balanced polynomials and combinatorial integration for the symmetric group S_d. Key features distinguishing our approach include:

- **Diverse ensemble support**: Unified integration over compact groups, circular ensembles, Gaussian ensembles (GUE, GOE, GSE), random pure states, diagonal unitary matrices (torus group), and unitary t-designs.
- **Symbolic dimensions and asymptotics**: Fully symbolic treatment of the dimension d, enabling the derivation of exact asymptotic expansions and universal scaling laws via automated Laurent series.
- **Symbolic trace logic**: A high-level interface that allows users to integrate trace polynomials directly, with the software automatically handling the reduction to Weingarten graphs.
- **Tensor network integration**: A native integration with `ITensors.jl` [Fishman et al. 2022], enabling the symbolic averaging of large-scale random tensor networks.
- **Partial trace**: A built-in function for computing partial traces of symbolic subsystems.
- **Performance**: Leveraging Julia's just-in-time compilation and efficient caching strategies to handle high-degree moments.

The remainder of this paper is organized as follows. Section 2 reviews the mathematical background of Haar integration. Section 3 describes the software architecture and core features. Section 4 provides usage examples, and Section 5 details the implementation. We present benchmarks in Section 6 and conclude in Section 7.

2 Background: Weingarten calculus

The core technique behind `IntegrateUnitary.jl` is the Weingarten calculus. For the compact matrix groups considered here, the integral of a polynomial in matrix elements can be expressed as a sum over a combinatorial set: permutations for the unitary case and pair partitions for the orthogonal and symplectic cases.

2.1 The unitary group $U(d)$

For the unitary group $U(d)$, the integral of a polynomial of degree k in both U and $\bar{U}$ is given by:

$$\int_{U(d)} U_{i_1 j_1} \dots U_{i_k j_k} \bar{U}_{i'_1 j'_1} \dots \bar{U}_{i'_k j'_k} dU = \sum_{\sigma,\tau \in S_k} \delta_\sigma(\vec{i}, \vec{i}') \delta_\tau(\vec{j}, \vec{j}') \mathrm{Wg}^U(\sigma\tau^{-1}, d), \tag{1}$$

where S_k is the symmetric group of degree k, and $\delta_\sigma(\vec{i}, \vec{i}')$ is a product of Kronecker deltas $\prod_m \delta_{i_m, i'_{\sigma(m)}}$. The Weingarten function $\mathrm{Wg}^U(\sigma, d)$ depends only on the cycle type of σ and, for generic d, can be expressed via the irreducible characters χ_λ of S_k:

$$\mathrm{Wg}^U(\sigma, d) = \frac{1}{(k!)^2} \sum_{\lambda \vdash k} \frac{(f^\lambda)^2 \chi_\lambda(\sigma)}{s_\lambda(1^d)}, \tag{2}$$

where f^λ is the dimension of the irreducible representation λ and $s_\lambda(1^d) = s_\lambda(1, \dots, 1)$ is the Schur polynomial evaluated at d variables all equal to one, representing the dimension of the corresponding $U(d)$ representation.

Crucially, $s_\lambda(1^d)$ is a polynomial in d, which implies that $\mathrm{Wg}^U(\sigma, d)$ is a rational function of d. This property is essential for symbolic integration, as it allows for exact results even when d is treated as a variable. Possible poles occur at integers d such that $|d| < k$, reflecting the breakdown of the expansion for small dimensions.

For fixed k and σ, in the asymptotic limit $d \to \infty$, the Weingarten function satisfies:

$$\mathrm{Wg}^U(\sigma, d) = O(d^{-k-|\sigma|}), \tag{3}$$

where $|\sigma|$ denotes the minimum number of transpositions required to generate σ, related to the number of cycles $c(\sigma)$ by $|\sigma| = k - c(\sigma)$. For the identity permutation, the leading order term is $\mathrm{Wg}^U(\mathrm{id}, d) \approx d^{-k}$, recovering the normalization expected from independent Gaussian entries at the naive limit.

Symbolic d pitfalls. While Weingarten functions are rational in d, they contain poles at small integer dimensions (typically $d < k$ for degree k moments). Furthermore, substituting numeric values can yield $0/0$ expressions, which are removable singularities, at specific points like $d = 1, 2$. `IntegrateUnitary.jl`'s `evaluate` function automatically simplifies these expressions to resolve such singularities. An important exception is pure trace moments $|\mathrm{tr}(U)|^{2k}$, whose exact value $\sum_{\lambda \vdash k, \ell(\lambda) \le d} (f^\lambda)^2$ depends on d as a step function rather than a rational function; these require a concrete integer dimension and raise an error for symbolic d.

Special Unitary group $SU(d)$. For all currently supported "balanced" polynomial expressions (where the number of U and $\bar{U}$ factors are equal), the integration over $SU(d)$ is equivalent to $U(d)$. Non-stable-range effects involving ϵ-tensor contractions for specific small d are not currently covered.

2.2 Orthogonal and symplectic groups

For the orthogonal group $O(d)$ and compact symplectic group $Sp(d)$ (in the package convention, a $d \times d$ matrix group with even $d = 2n$), odd moments vanish and even moments are indexed by the set of pair partitions P_{2k} of $\{1, \dots, 2k\}$.

For $O(d)$, the integral is:

$$\int_{O(d)} O_{i_1 j_1} \dots O_{i_{2k} j_{2k}} dO = \sum_{p,q \in P_{2k}} \delta_p(\vec{i}) \delta_q(\vec{j}) \mathrm{Wg}^O(p, q, d). \tag{4}$$

Here $\delta_p(\vec{i}) = \prod_{\{a,b\} \in p} \delta_{i_a i_b}$. Similarly, for the symplectic group $Sp(d)$, with standard symplectic form $J = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$, the formula is:

$$\int_{Sp(d)} S_{i_1 j_1} \dots S_{i_{2k} j_{2k}} dS = \sum_{p,q \in P_{2k}} \Delta_p^J(\vec{i})\, \Delta_q^J(\vec{j})\, \mathrm{Wg}^{Sp}(p, q, d), \tag{5}$$

where $\Delta_p^J(\vec{i}) = \prod_{(a,b) \in p,\ a<b} J_{i_a i_b}$ contracts each pair through the symplectic form J, introducing signs ± 1. For detailed closure relations on these groups, see Collins and Śniady [Collins and Śniady 2006].

`IntegrateUnitary.jl` leverages the fundamental duality between the orthogonal and symplectic Weingarten functions:

$$\mathrm{Wg}^{Sp}(p, q, d) = (-1)^{\mathrm{loops}(p,q)} \mathrm{Wg}^O(p, q, -d). \tag{6}$$

where $\mathrm{loops}(p, q)$ is the number of loops in the graph formed by the union of the two pair partitions p and q. This relation allows the package to unify the codebase, computing symplectic integrals by reusing the optimized orthogonal engine with an appropriately signed dimension parameter $d \to -d$.

Both Wg^O and Wg^{Sp} are rational functions of d, and their poles mark singularities of the rational inverse-Gram representation; the Haar integrals themselves are well defined for admissible group dimensions.

`IntegrateUnitary.jl` efficiently computes these functions using character-based formulas and Gram-matrix inversion (detailed in Section 5), caching results to accelerate repeated integration tasks.

2.3 Permutation group S_d

We represent the symmetric group S_d by $d \times d$ permutation matrices and integrate with respect to the normalized counting measure. For a monomial in the matrix entries $P_{i,j}$, repeated identical factors are first collapsed (equivalently, we work with the set of distinct index pairs). The integral is non-zero only if this deduplicated set is consistent with a permutation (i.e., all row indices are distinct and all column indices are distinct). The result is:

$$\int_{S_d} P_{i_1 j_1} \dots P_{i_k j_k} dP = \frac{(d-k)!}{d!}, \tag{7}$$

where k is the number of distinct index pairs after the collapse.

3 Software features

`IntegrateUnitary.jl` is built on top of the `Symbolics.jl` ecosystem [Gowda et al. 2022], providing a seamless experience for Julia users.

3.1 Groups and measures

The package supports a wide range of integration measures, grouped as follows:

- **Compact groups**: Standard Haar measures on the unitary $U(d)$, special unitary $SU(d)$, orthogonal $O(d)$, and symplectic $Sp(d)$ groups. These are defined via `dU(d)`, `dSU(d)`, `dO(d)`, and `dSp(d)`, where the argument is the matrix dimension d; for concrete dimensions, `dSp(d)` requires even d. For balanced polynomials in the currently

supported stable range, $SU(d)$ and $U(d)$ integrals coincide; non-stable-range ϵ-tensor effects at specific small d are not currently covered.

- **Circular ensembles**: Measures for the circular unitary (CUE), orthogonal (COE), and symplectic (CSE) ensembles, accessed via `dCUE(d)`, `dCOE(d)`, and `dCSE(d)`. The constructor `dCUE(d)` is an alias for `dU(d)`, while `dCSE(d)` requires even concrete matrix dimensions; COE and CSE represent symmetric and self-dual unitary matrices, respectively.
- **Quantum states**: The Fubini–Study measure on pure states $|\psi\rangle$ drawn from the complex projective space $\mathbb{C}P^{d-1}$ [Bengtsson and Życzkowski 2017; Życzkowski and Sommers 2001], accessed via `dPsi(d)`. This corresponds to the distribution of the first column of a Haar-random unitary matrix and is represented as a $(d, 1)$ symbolic matrix.
- **Permutation groups**: Measures for the Symmetric Group S_d (`dPerm(d)`) and the ensemble of centered permutation matrices (`dCPerm(d)`). Centered permutations Y satisfy $Y_{ij} = P_{ij} - 1/d$, where $P \in S_d$.
- **Gaussian and Ginibre ensembles**: Support for the Gaussian unitary (GUE), orthogonal (GOE), and symplectic (GSE) ensembles, as well as the complex (GinUE), real (GinOE), and symplectic (GinSE) Ginibre ensembles, accessed via `dGUE(d)`, `dGOE(d)`, `dGSE(d)`, `dGinUE(d)`, `dGinOE(d)`, and `dGinSE(d)`. For concrete dimensions, `dGSE(d)` and `dGinSE(d)` require even d. These rely on Wick's theorem [Wick 1950] for integration rather than Weingarten calculus.
- **Unitary designs**: Unitary t-designs (`dDesign(d, t)`), which mimic the first t moments of the Haar measure, useful for studying pseudo-randomness in quantum circuits.
- **Stiefel manifolds**: The Stiefel manifold $V_k(\mathbb{C}^d)$, representing $d \times k$ matrices with orthonormal columns. This generalizes Haar-random pure states ($k = 1$) and is implemented via `dStiefel(d, k)`, with $k \leq d$ required for concrete dimensions.
- **Diagonal unitary matrices**: Integration over the torus group T^d (`dDiagUnitary(d)`), representing independent phase averaging for each diagonal entry.
- **Matrix integration**: Native support for integrating AbstractArray-valued expressions with concrete integer result dimensions, enabling coordinate-free validation of matrix identities (e.g., $\mathbb{E}[UU^\dagger] = I$).

3.2 Harish-Chandra-Itzykson-Zuber (HCIZ) integrals

Beyond polynomial moments, `IntegrateUnitary.jl` implements the closed-form Harish-Chandra-Itzykson-Zuber (HCIZ) integral [Harish-Chandra 1958; Itzykson and Zuber 1980], a fundamental object in random matrix theory related to the character expansion of the exponential function:

$$\int_{U(d)} dU e^{\mathrm{Tr}(AUBU^\dagger)} = \left(\prod_{p=1}^{d-1} p!\right) \frac{\det(e^{a_i b_j})_{i,j=1}^{d}}{\Delta(a)\Delta(b)}, \tag{8}$$

where a_i and b_j are the eigenvalues of the source matrices A and B, and $\Delta(x)$ denotes the Vandermonde determinant. When the eigenvalues are symbolic and non-degenerate, the evaluation is exact; for floating-point input the result is numerical.

`IntegrateUnitary.jl` provides two primary interfaces for these integrals. The **eigenvalue interface** allows users to pass vectors of eigenvalues directly, facilitating purely symbolic derivations where the spectrum is defined algebraically. The **matrix interface** accepts Hermitian matrices A and B and dispatches on the element type. For numeric matrices, eigenvalues are computed via standard diagonalization (`eigen`). For symbolic matrices of type `Matrix{Num}`, the package

extracts eigenvalues from diagonal matrices of any dimension and from general 2×2 matrices via the quadratic formula; larger non-diagonal symbolic matrices require the user to supply eigenvalues directly. For `SymbolicMatrix` inputs with a concrete integer dimension, where explicit entries are not available, the package introduces formal eigenvalue symbols $(a_1, \ldots, a_d, b_1, \ldots, b_d)$ and evaluates the formula in terms of these symbols; symbolic dimensions are not supported here because the formula requires constructing a finite set of eigenvalue symbols and a $d \times d$ determinant. When degenerate eigenvalues are detected in numeric input, the implementation first sorts both spectra (using a total order by real then imaginary part) to make the procedure permutation-invariant, and then applies independent perturbations $a_i \to a_i + i\,\epsilon_a$ and $b_i \to b_i + i\,\epsilon_b$, where $\epsilon_a = \max(\|a\|_\infty, 1) \cdot 10^{-12}$ and $\epsilon_b = \max(\|b\|_\infty, 1) \cdot 10^{-12}$. This breaks exact degeneracies in both Vandermonde denominators while introducing errors of order $O(\max(\epsilon_a, \epsilon_b))$ in the result, which is negligible for double-precision arithmetic. For symbolic eigenvalues, degenerate cases must be resolved analytically by the user (e.g., via L'Hôpital's rule).

3.3 Gaussian random matrix ensembles

In addition to compact groups, `IntegrateUnitary.jl` supports integration over the Gaussian ensembles (GUE, GOE, GSE). These are implemented using Wick's theorem [Wick 1950] (Isserlis' theorem) for moment contraction. For a Gaussian Hermitian matrix H, the expectation values are determined by the pairwise contractions:

- **GUE**: $\langle H_{ij}\bar{H}_{kl}\rangle = \delta_{ik}\delta_{jl}$, leading to $\langle \mathrm{tr}(H^2)\rangle = d^2$, $\langle \mathrm{tr}(H^4)\rangle = 2d^3 + d$, and $\langle \mathrm{tr}(H^6)\rangle = 5d^4 + 10d^2$.
- **GOE**: $\langle H_{ij}H_{kl}\rangle = \delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}$ (for real symmetric H), leading to $\langle \mathrm{tr}(H^2)\rangle = d^2 + d$.
- **GSE**: For self-dual Hermitian matrices, the package utilizes the even-moment duality relation:

$$\langle \mathrm{tr}(H^k)\rangle_{\mathrm{GSE}}(d) = (-1)^{\frac{k}{2}+1}\langle \mathrm{tr}(H^k)\rangle_{\mathrm{GOE}}(-d), \qquad k \in 2\mathbb{N}, \tag{9}$$

  while odd moments vanish, $\langle \mathrm{tr}(H^{2m+1})\rangle_{\mathrm{GSE}} = 0$. This implies $\langle \mathrm{tr}(H^2)\rangle = d^2 - d$.

This approach allows `IntegrateUnitary.jl` to compute moments for all three ensembles while maintaining full support for symbolic dimensions d. For concrete integer dimensions, the symplectic measure constructor enforces even size: `dGSE(n)` raises an error when `n` is odd.

3.4 Ginibre ensembles

Moving beyond Hermitian matrices, `IntegrateUnitary.jl` supports the Ginibre ensembles (GinUE, GinOE, GinSE) [Ginibre 1965], which consist of non-Hermitian matrices with independent and identically distributed (i.i.d.) Gaussian entries. These are integrated using similar Wick contraction rules:

- **GinUE**: $\langle G_{ij}\bar{G}_{kl}\rangle = \delta_{ik}\delta_{jl}$, leading to $\langle \mathrm{tr}(GG^\dagger)\rangle = d^2$, $\langle \mathrm{tr}(GG^\dagger)^2\rangle = d^4 + d^2$, and $\langle \mathrm{tr}((GG^\dagger)^2)\rangle = 2d^3$. Only contractions between G and its conjugate $\bar{G}$ are non-vanishing.
- **GinOE**: $\langle G_{ij}G_{kl}\rangle = \delta_{ik}\delta_{jl}$ for real entries.
- **GinSE**: Symplectic entries treated via duality relations. For concrete integer dimensions, `dGinSE(n)` requires even `n` and raises an error on odd `n`.

Like the beta ensembles, these rules are implemented to support both symbolic dimensions d and high-level trace expressions.

3.5 Circular ensembles

`IntegrateUnitary.jl` also supports integration over the circular ensembles of random unitary matrices, which are important in the study of quantum chaos and symmetric spaces.

- **CUE (Circular unitary ensemble)**: This corresponds to the standard Haar measure on $U(d)$. The matrices are unitary without further symmetry constraints.
- **COE (Circular orthogonal ensemble)**: Consists of symmetric unitary matrices ($S = S^T$). These are constructed as $S = UU^T$ where U is Haar-distributed on $U(d)$. `IntegrateUnitary.jl` handles integrals over COE by mapping the moments of S to higher-order moments of U, reducing the problem to standard unitary Weingarten calculus.
- **CSE (Circular symplectic ensemble)**: Consists of self-dual unitary matrices ($S = UU^R$, where $U^R = JU^TJ^T$ is the dual transpose). The integration is similarly performed by mapping to $U(d)$ integrals (with d even).

These are accessed via `dCOE(d)`, `dCSE(d)`, and `dCUE(d)` (alias for `dU`).

3.6 Unitary designs: Finite-moment measures and guards

Unitary t-designs [Ambainis and Emerson 2007; Dankert et al. 2009; Gross et al. 2007] are ensembles of unitary matrices that mimic the Haar measure up to the t^{th} moment. Specifically, a set of unitaries $\mathcal{D} \subset U(d)$ is a t-design if:

$$\mathbb{E}_{U\in\mathcal{D}}[P(U,\bar{U})] = \int_{U(d)} P(U,\bar{U})dU \tag{10}$$

for all polynomials P of degree $q \le t$ in the entries of U and $\bar{U}$.

`IntegrateUnitary.jl` provides the `dDesign(d, t)` measure to represent such ensembles. The implementation enforces the moment-matching condition as follows:

(1) For balanced integrands (equal degree q in U and $\bar{U}$) with $q \le t$, the package returns the exact Haar-averaged result using standard Weingarten calculus.
(2) For balanced integrands with $q > t$, the package raises an explicit error, preventing incorrect assumptions about the design's higher-moment behavior.
(3) Unbalanced integrands (different degree in U and $\bar{U}$) return zero, which is the Haar-correct value. This is guaranteed correct for total degree $\le t$; for higher-degree unbalanced monomials, a t-design does not guarantee this value, but no error is currently raised.

This feature allows researchers to verify whether specific quantum protocols or randomized benchmarking schemes rely only on the t-moment properties of the sampling distribution.

3.7 Partial trace

The package includes a `partial_trace(M, dims, subsystem)` function for quantum information tasks. It computes the partial trace of a matrix M over a specified subsystem, where the subsystem dimensions (e.g., d_A, d_B with $d = d_Ad_B$) are concrete integers. The matrix entries themselves may be symbolic expressions, for instance, products of `SymbolicMatrix` elements, so the resulting reduced density matrix retains symbolic dependencies suitable for further Haar integration. This enables the calculation of entanglement metrics such as purity by integrating the trace of the squared reduced density matrix, as demonstrated in Section 6.

3.8 Symbolic dimensions and asymptotics

A standout feature of `IntegrateUnitary.jl` is its deep support for symbolic dimensions d. By leveraging the polynomial nature of group characters and Schur functions, `IntegrateUnitary.jl` computes many supported element-wise and trace-of-product integrals as exact rational functions of d; for supported Haar/Weingarten-type matrix-group integrals, these take the form:

$$\int_{G(d)} P(U, \bar{U}) dU = \frac{N(d)}{D(d)}, \tag{11}$$

where $N(d)$ and $D(d)$ are polynomials. (Not all results are rational: pure trace moments $|\mathrm{tr}(U)|^{2k}$ depend on d as a step function and require a concrete integer dimension; see Section 3.12.) This capability is critical for studying:

- **Transition points**: Identifying poles in $D(d)$ that mark singularities of the rational representation; at admissible integer dimensions, removable singularities may still have well-defined values.
- **Thermodynamic limits**: The `asymptotic(expr, measure, order)` function computes the series expansion of the result in powers of $1/d$. This automates the extraction of large-d behavior for rational-in-d observables (for example, high-degree entry moments and trace-polynomial integrals), where exact expressions can be combinatorially heavy. Pure trace moments $|\mathrm{tr}(U)|^{2k}$ are outside this workflow: they should be evaluated with `integrate(..., dU(n))` at concrete integer n.

Even when the input dimension is numeric (e.g., $d = 3$), for measures supporting symbolic reconstruction, the `asymptotic` routine can introduce a dummy symbolic variable to perform the expansion for these rational-in-d observables, providing theoretical insights alongside numerical results.

3.9 Symbolic trace logic

To simplify the integration of high-rank tensor networks and complex trace expressions, `IntegrateUnitary.jl` introduces a **symbolic trace logic** system. This system abstracts away explicit tensor indices, allowing users to define computations in terms of coordinate-free matrix objects:

- **Symbolic Matrix**: The `SymbolicMatrix` type represents an operator tagged by its role, such as a Haar unitary U, its adjoint $U^\dagger$, or a constant matrix M.
- **Lazy Evaluation**: Products such as $A * B$ build `SymbolicMatrixProduct` objects, while trace operations (`tr` or `tr_lazy`) generate `LazyTrace` objects. These objects maintain algebraic structures (products of traces of matrix strings) without expanding indices, e.g., representing $\mathrm{tr}(UAU^\dagger B)$ as a graph cycle rather than a sum over four indices.
- **Graph Reconstruction**: During integration, the package interprets these lazy structures as contraction graphs. It automatically reconstructs the connectivity required for the relevant permutation, pair-partition, or Wick contractions, effectively converting the "index-free" input into the precise tensor contractions required by the relevant Weingarten or Wick formulas.

This abstraction significantly reduces the potential for index-mismatch errors and renders the code nearly isomorphic to the mathematical pen-and-paper formulation of the problem.

3.10 ITensors.jl integration

Modern quantum circuit analysis and holographic duality models often rely on tensor network representations. `IntegrateUnitary.jl` provides a specialized extension for `ITensors.jl` [Fishman et al. 2022], allowing users to integrate entire networks of `ITensor` objects symbolically.

To integrate an ITensor network, the user marks the random unitaries using the `ITensorUnitary` wrapper, which specifies the tensor's input and output indices. This explicit marking prevents ambiguities that arise from carrying index tags (e.g., "Haar") over to constant tensors. The integration routine then analyzes the contraction topology of the network and expands it into a sum of deterministically contracted ITensors, each weighted by the appropriate Weingarten function. This graphical Weingarten engine is significant for its automatic handling of large, sparse networks where manual index expansion would be computationally prohibitive.

3.11 Comparison with existing tools

Table 1 provides a detailed feature comparison of `IntegrateUnitary.jl` with the two most closely related packages: `RTNI` [Fukuda et al. 2019], a Mathematica package for integrating Haar-random tensor networks, and `Haarpy` [Cardin et al. 2024], a Python library for Weingarten calculus over classical compact groups. The comparison highlights three key distinctions:

(1) **Ensemble breadth**: Among the compared tools, `IntegrateUnitary.jl` supports Gaussian, Ginibre, and circular ensembles alongside compact groups, as well as discrete structures such as permutation groups, unitary t-designs, Stiefel manifolds, and diagonal unitaries.
(2) **Symbolic abstractions**: Neither `RTNI` nor `Haarpy` provides asymptotic $1/d$ expansions or `IntegrateUnitary.jl`-style symbolic trace logic. `Haarpy` does not provide matrix-valued integration, while `IntegrateUnitary.jl` provides direct matrix-valued integration for concrete-size array outputs and `RTNI` exposes graph/tensor outputs that typically require additional trace or scalarization post-processing for like-for-like scalar comparisons. These features significantly reduce the manual effort required for common calculations.
(3) **Platform**: `RTNI` requires the proprietary Mathematica engine (though a Python port exists), while `Haarpy` and `IntegrateUnitary.jl` are fully open-source. `IntegrateUnitary.jl`'s Julia implementation offers JIT compilation and native type specialization, providing performance advantages for large-scale symbolic computations. Direct timing comparisons with `Haarpy` and `RTNI` are presented in Sections 6.4 and 6.5.

3.12 Limitations and scope

The current implementation has several known limitations:

- $SU(d)$ **scope**: Integration over $SU(d)$ is currently supported only for *balanced* polynomial expressions, i.e., those containing equal numbers of U and $\bar{U}$ factors. For such expressions, $SU(d)$ and $U(d)$ integrals coincide. Non-balanced integrals, which require ϵ-tensor contractions and exhibit dimension-dependent behavior specific to small d, are not yet supported. In the current backend, these non-balanced queries are evaluated through the $U(d)$ rule and therefore return zero.
- **HCIZ degeneracies**: The Harish-Chandra-Itzykson-Zuber integral formula involves Vandermonde determinants in the denominator, which vanish when eigenvalues coincide. For numeric inputs with degenerate spectra, the package applies a small perturbation (see Section 3); for symbolic inputs, degenerate cases currently require manual handling via L'Hôpital's rule or limiting procedures.

Table 1. Feature comparison of symbolic Haar integration packages. ✓ = fully supported, $\sim^{a}/\sim^{b}$ = partial support with caveat [a]/[b], – = not supported.

Category	Feature	`IntegrateUnitary.jl`	`RTNI`	`Haarpy`
Groups	$U(d)$	✓	✓	✓
	$SU(d)$ (balanced)	✓	–	–
	$O(d)$	✓	–	✓
	$Sp(d)$	✓	–	$\sim^{a}$
Ensembles	Circular (COE/CSE)	✓	–	$\sim^{a}$
	Gaussian (GUE/GOE/GSE)	✓	–	–
	Ginibre (GinUE/GinOE/GinSE)	✓	–	–
Discrete/other	Permutation groups	✓	–	✓
	Unitary t-designs	✓	–	–
	Stiefel manifolds	✓	–	–
	Diagonal unitaries	✓	–	–
	Random pure states	✓	–	–
Symbolic	Symbolic dimension d	✓	✓	✓
	Asymptotic $1/d$ expansions	✓	–	–
	Symbolic trace logic	✓	–	–
Interfaces	Tensor network integration	✓	✓	–
	Matrix-valued integration	✓	$\sim^{b}$	–
	HCIZ integrals	✓	–	–
Platform	Language	Julia	Mathematica/Python	Python
	Open-source runtime	✓	–	✓
	License	Apache 2.0	GPL v3.0	Apache 2.0

[a]Haarpy: Weingarten functions available but full integration not yet implemented (Sp(d), CSE).

[b]RTNI: graph/tensor-network outputs are available, but direct scalar value comparison typically requires extra trace/scalarization post-processing.

- **Polynomial degree**: Due to the factorial growth of the symmetric group ($|S_k| = k!$) and pair partitions ($|M_{2k}| = (2k-1)!!$), exact symbolic integration is practically limited to polynomial degrees $2k \lesssim 12$ on standard hardware.
- **Trace moments**: Pure trace moments $|\mathrm{tr}(U)|^{2k}$ depend on d as a step function (not a polynomial), so they require a concrete integer dimension. For integer d, the library returns the exact value; for symbolic d, an `ArgumentError` is raised.
- **Matrix-valued integration**: Direct matrix-valued integration of `SymbolicMatrix` and `SymbolicMatrixProduct` expressions requires concrete integer result dimensions. If the output size is symbolic, users should scalarize the expression (for example with `tr(...)`) instead of requesting a full matrix-valued result.

4 Usage examples

We illustrate the capabilities of `IntegrateUnitary.jl` with several examples, ranging from basic component-wise integration to high-level symbolic trace manipulations and asymptotic expansions. These examples demonstrate how the package's design principles facilitate practical research tasks.

4.1 Basic integration

Consider the partial moment $\int_{U(d)} |U_{11}|^2 dU$. This integral represents the average magnitude squared of a single entry of a Haar-random unitary matrix. In `IntegrateUnitary.jl`, this can be computed directly with `@integrate`: the macro infers both the symbolic matrix symbol (`U`) and the symbolic dimension (`d`) from the measure context `dU(d)`:

```
using IntegrateUnitary
# The @integrate macro infers U and d from the measure context
result = @integrate abs(U[1,1])^2 dU(d)
# Output: 1/d

# Complex trace moments require a concrete integer dimension
res_tr = @integrate abs(tr(U))^4 dU(10)
# Output: 2

# Mixed-index fourth moment
res_c = @integrate U[1, 1] * conj(U[1, 2]) * U[2, 2] * conj(U[2, 1]) dU(d)
# Output: -1/(d*(d^2 - 1))

# Special Unitary group (stable range): matches U(d) for balanced moments
@integrate abs(U[1, 1])^2 dSU(d)
# Output: 1/d
```

The result $1/d$ confirms the expected normalization: since the rows and columns of a unitary matrix are unit vectors, the average squared magnitude of any entry must be $1/d$.

4.2 Circular ensembles

For the circular ensembles, we can observe distinct statistical properties compared to the standard Haar measure. Here, we compute the second moment of a diagonal entry for the symmetric (COE) and self-dual (CSE) cases:

```
# Circular Orthogonal Ensemble (S is symmetric)
@integrate abs(S[1, 1])^2 dCOE(d)
# Output: 2/(d+1)

# Circular Symplectic Ensemble (S is self-dual)
@integrate abs(S[1, 1])^2 dCSE(d)
# Output: 1/(d-1)

# Orthogonal group O(d): real matrix entries
@integrate O[1, 1]^2 dO(d)
# Output: 1/d

@integrate O[1, 1]^4 dO(d)
```

```
# Output: 3 / (d*(d+2))

# Symplectic group Sp(d): mixed moments
@integrate abs(Sp[1,1])^2 * abs(Sp[1,2])^2 dSp(d)
# Output: 1 / (d + d^2)   (poles at d = 0, -1)
```

These results contrast with the Haar unitary value of $1/d$, reflecting the constraints imposed by the respective symmetries ($S = S^T$ for COE, $S = S^R$ for CSE, $O^T O = I$ for orthogonal, and the symplectic form for $Sp(d)$).

4.3 Matrix integration

`IntegrateUnitary.jl` allows for the direct integration of matrix-valued expressions when the result dimensions are concrete integers. This eliminates the need for manual element-wise iteration or broadcasting, automatically performing the integration over all array entries. For Haar-random unitary matrices, we verify the relation $\int UU^\dagger dU = I$:

```
using IntegrateUnitary
# Integrate the matrix product U * U'
# IntegrateUnitary handles the element-wise operations internally
@integrate U * U' dU(2)
# Output: Identity Matrix I
```

4.4 Symbolic trace logic

For more complex expressions involving traces of random matrices, manual index contraction is error-prone. The symbolic trace interface simplifies the workflow by allowing users to define the calculation in coordinate-free notation. Here, we compute $\int \mathrm{tr}(UAU^\dagger B)dU$, a standard integral appearing in the study of quantum channels (specifically, the definition of the depolarizing channel):

```
# Compute integral of tr(U A U' B)
# The @integrate macro automatically treats unknown symbols as constant matrices
@integrate tr(U * A * U' * B) dU(d)
# Output: tr(A)tr(B)/d
```

This result neatly recovers the identity $\mathbb{E}[UAU^\dagger] = \mathrm{tr}(A)\mathbb{I}/d$.

4.5 Asymptotic expansions

Analyzing behavior in the large-d limit is crucial for understanding thermodynamic properties and concentration of measure. `IntegrateUnitary.jl` facilitates this analysis by expanding exact rational results into power series. Consider the fourth moment of a matrix entry, $|U_{11}|^4$:

```
using IntegrateUnitary, Symbolics
@variables d
```

```
# The shorthand factory functions can be used for explicit variable creation
U = symbolic_unitary(:U, d)
expr = abs(U[1,1])^4
# Exact result computed internally: 2/(d*(d+1))
asymp_res = asymptotic(expr, dU(d), 4)
# Output: 2/d^2 - 2/d^3 + 2/d^4
```

This facility is particularly valuable for extracting universal scaling laws from combinatorially complex exact expressions. For higher trace-polynomial observables, it directly exposes both leading behavior and finite-size corrections in powers of $1/d$:

```
using IntegrateUnitary, Symbolics
@variables d
U = symbolic_unitary(:U, d)
A = SymbolicMatrix(:A); B = SymbolicMatrix(:B)
C = SymbolicMatrix(:C); D = SymbolicMatrix(:D)

asymp_tp = asymptotic(tr(U*A*U'*B*U*C*U'*D), dU(d), 3)
# Leading terms:
# (tr(A)tr(B*D)tr(C) + tr(A*C)tr(B)tr(D))/d^2
# - (tr(A)tr(B)tr(C)tr(D) + tr(A*C)tr(B*D))/d^3
```

In contrast, pure trace moments have exact finite-d dependence given by a step function in integer d:

$$\mathbb{E}\left[|\mathrm{tr}(U)|^{2k}\right] = \sum_{\lambda \vdash k,\ \ell(\lambda) \le d} (f^\lambda)^2.$$

Hence, for fixed k, the value equals $k!$ exactly once $d \ge k$ (for example, $\mathbb{E}[|\mathrm{tr}(U)|^6] = 6$ for all integer $d \ge 3$), and there is no $1/d$-type subleading correction series in that stabilized regime.

As a physically grounded application, consider Page's result on the average entanglement of random bipartite states [Page 1993]. For a random pure state on $\mathbb{C}^n \otimes \mathbb{C}^n$, the average purity of either subsystem is $\mathbb{E}[\mathrm{tr}(\rho_A^2)] = 2n/(n^2+1)$. The `asymptotic` function accepts any rational expression, so we can immediately extract the large-n behavior:

```
using IntegrateUnitary, Symbolics
@variables n
page_purity = 2n / (n^2 + 1)
asymptotic(page_purity, n, 5)
# Output: 2/n - 2/n^3 + 2/n^5
```

The leading term $2/n$ shows that the subsystem purity is of order $1/n$ as the dimension grows. For comparison, a maximally mixed n-dimensional state has purity exactly $1/n$, so Page's law indicates highly mixed (and thus nearly maximally entangled) reduced states rather than equality with the maximally mixed value. The subleading corrections $-2/n^3 + \cdots$ quantify the finite-size deviations from maximal entanglement. Such expansions automate the derivation of scaling laws where manual simplification of the underlying Weingarten sums would be prohibitive.

4.6 Tensor network integration

Finally, we demonstrate the integration of a tensor network using the `ITensors.jl` interface. This approach is particularly useful for Haar averaging of random quantum circuits and tensor networks without explicit index manipulation. We define a balanced network consisting of a random unitary U, its adjoint $U^\dagger$, and constant tensors A, B, then compute its Haar average.

```
using IntegrateUnitary, ITensors
i, j, k, l = Index(2), Index(2), Index(2), Index(2)

# Define U and its adjoint U_dag
U = ITensorUnitary(out_indices=[i], in_indices=[j])
U_dag = ITensorUnitary(out_indices=[k], in_indices=[l], is_adj=true)

# Constant tensors A and B to form a trace-like network
A = randomITensor(j, k); B = randomITensor(l, i)

# Integrate the balanced network [U, A, U_dag, B] over U(2)
integrate([U, A, U_dag, B], dU(2))
# result is a scalar ITensor from Weingarten delta contractions
# (for d=2 the overall prefactor is 1/2)
```

This high-level approach allows symbolic integration to be expressed directly at the network level, avoiding manual elementwise index expansion while letting the underlying engine operate on the network topology.

4.7 Permutation groups

Symbolic integration over the Symmetric Group S_d is particularly useful for combinatorial problems and studying centered permutation ensembles.

```
# Average of a product of entries
@integrate P[1, 1] * P[2, 2] dPerm(d)
# Output: 1 / (d * (d - 1))

# Centered permutations Y = P - J/d
@integrate Y[1, 1]^2 dCPerm(d)
# Output: (d - 1) / d^2
```

The first result, $1/(d(d-1))$, is the probability that a uniformly random permutation maps both 1 and 2 to themselves, reflecting the well-known inclusion-exclusion counting of derangements. The second result shows that centring the permutation matrix removes the $1/d$ mean, leaving a variance that converges to $1/d$ for large d.

4.8 Diagonal unitary matrices

For the group of diagonal unitary matrices (the torus T^d), integration reduces to independent phase averaging. This arises naturally in the dephasing channel and in studies of quantum coherence.

```
# Average of |D_11|^2
@integrate abs(D[1, 1])^2 dDiagUnitary(d)
# Output: 1
```

The result 1 reflects the fact that each diagonal entry $D_{ii} = e^{i\theta_i}$ has unit modulus, so $|D_{11}|^2 = 1$ deterministically.

4.9 Stiefel manifolds

Integration over the Stiefel manifold $V_k(\mathbb{C}^d)$ is supported, generalizing pure states to higher-rank orthonormal frames. This is crucial for variational quantum algorithms and tensor network optimization [Edelman et al. 1998].

```
# E[|V_{1,1}|^2]
@integrate abs(V[1, 1])^2 dStiefel(d, 2)
# Output: 1 / d
```

The result $1/d$ coincides with the unitary case because a random element of $V_2(\mathbb{C}^d)$ is simply the first two columns of a Haar-random $U(d)$ matrix, so entries share the same second-moment structure.

4.10 Ginibre ensembles

Integration over Ginibre ensembles is useful for studying non-Hermitian random matrix properties. We compute the second moment $\langle \mathrm{tr}(GG^\dagger) \rangle$:

```
# Compute <tr(G G')>
@integrate tr(G * G') dGinUE(d)
# Output: d^2
```

The result d^2 follows from the fact that each entry of a complex Ginibre matrix is an independent standard complex Gaussian, so $\mathbb{E}[|G_{ij}|^2] = 1$ and $\mathbb{E}[\mathrm{tr}(GG^\dagger)] = \sum_{i,j} \mathbb{E}[|G_{ij}|^2] = d^2$.

5 Implementation details

`IntegrateUnitary.jl` is written in pure Julia [Bezanson et al. 2017], utilizing the language's multiple dispatch and metaprogramming features to build a highly modular and extensible system. The core architecture consists of three main components: the integration pipeline, the Weingarten engine, and the symbolic trace logic. Figure 1 provides an overview of the data flow.

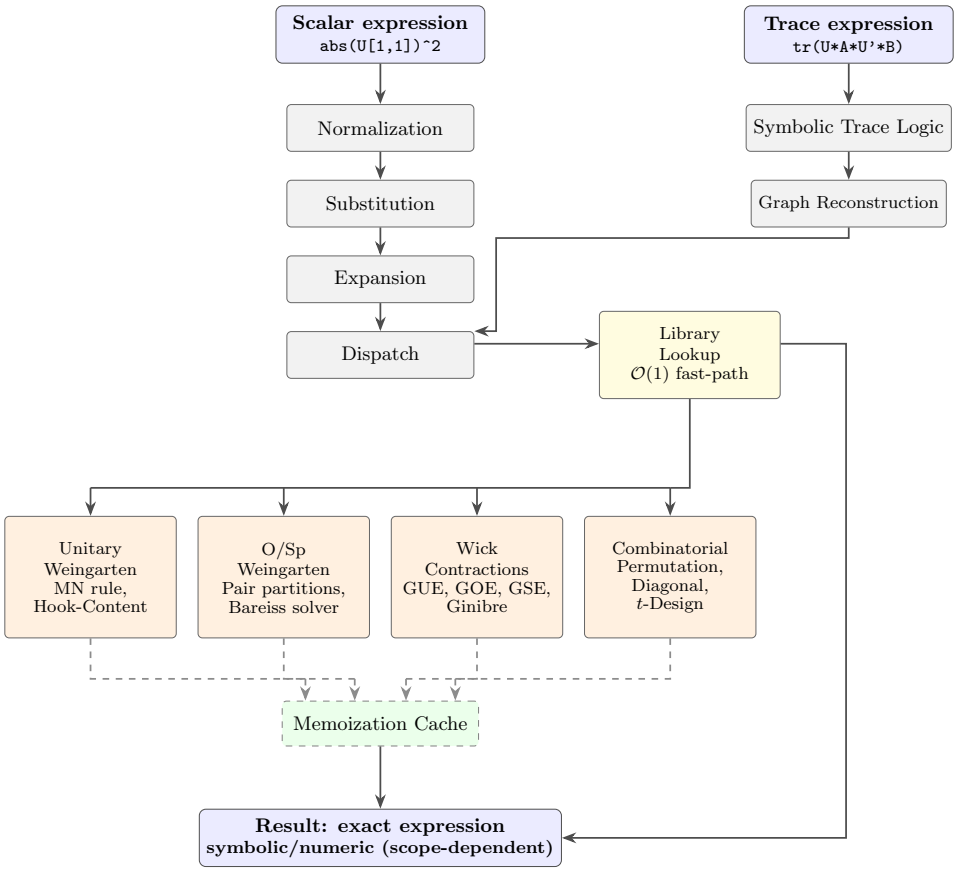

Fig. 1. Architecture of `IntegrateUnitary.jl`. Scalar expressions follow the left path through normalization, substitution, expansion, and dispatch into the library lookup. Trace expressions are first analyzed by the symbolic trace logic, which reconstructs the Weingarten graph before entering the same dispatch stage. Common patterns are returned directly by the library lookup in $O(1)$ time, while misses fall through to the specialized engines. Performance-critical internals (representation-theoretic quantities and Weingarten-related data) are memoized/cached for reuse.

5.1 Integration pipeline

The integration process follows a systematic pipeline designed to normalize and dispatch symbolic expressions efficiently:

(1) **Normalization**: The input expression is first normalized using a ruleset based on `SymbolicUtils.jl`. This step expands complex conjugates and rewrites composite functions, such as $\mathrm{abs}(z)^2 \to z\bar{z}$, $\mathrm{real}(z) \to \frac{1}{2}(z+\bar{z})$, and $\mathrm{imag}(z) \to \frac{1}{2i}(z-\bar{z})$, into polynomial forms amenable to integration.
(2) **Substitution**: Symbolic variables representing matrix elements U_{ij} are replaced by an internal atomic representation that facilitates pattern matching against the integration measure.
(3) **Expansion**: The expression is algebraically expanded into a sum of monomials. This relies on the efficient sparse polynomial handling of `Symbolics.jl`.

(4) **Dispatch**: Each monomial is analyzed to extract the indices of the random matrices. The integration is then dispatched to a centralized engine. This engine leverages the `AbstractMeasure` base type and a unified `measure_info` interface, which provides the necessary metadata (substitution dictionaries, dimension parameters, and ensemble tags) for all supported measures. This architecture eliminates redundant boilerplate and ensures consistent dispatch behavior across the library.

5.2 Weingarten engine

The calculation of Weingarten functions and combinatorial contractions is centralized within a generic integration engine. To achieve high performance and extensibility, `IntegrateUnitary.jl` implements specialized logic for different mathematical structures while sharing the top-level dispatch mechanism:

- **Unitary group**: Characters of the symmetric group are computed using the Murnaghan-Nakayama rule, which is significantly faster than determinantal formulas for sparse partitions. Dimensions of irreducible representations $s_\lambda(1^d)$ are computed via the Hook-Content formula:

$$s_\lambda(1^d) = \prod_{(i,j)\in\lambda} \frac{d + c_{i,j}}{h_{i,j}} \tag{12}$$

  where $c_{i,j} = j - i$ is the content and $h_{i,j}$ is the hook length. This formulation allows the package to handle symbolic dimensions natively in supported workflows, producing exact rational functions of d where applicable.
- **Orthogonal and symplectic groups**: For these groups, the package generates pair partitions recursively, canonicalizing them (typically as sorted pairs) to enable $O(1)$ lookup. For symbolic dimensions d, `IntegrateUnitary.jl` utilizes a specialized univariate polynomial solver based on the Bareiss algorithm [Bareiss 1968] with exact `BigInt` arithmetic. Furthermore, to ensure stability for high-degree Weingarten sums where traditional simplification can fail, the engine employs a custom exact rational summation logic. Integration over the symplectic group reuses this optimized engine via duality:

$$\mathrm{Wg}^{Sp}(p, q, d) = (-1)^{\mathrm{loops}(p,q)}\mathrm{Wg}^{O}(p, q, -d). \tag{13}$$

- **Gaussian and Ginibre Ensembles**: Integration over Gaussian ensembles is performed via Wick contractions. The engine decomposes the monomial into atomic factors and generates all valid pair partitions of indices, weighted by the ensemble-specific contraction rules.
- **Diagonal unitary matrices**: For the torus group T^d, the integration is performed by verifying that the integrand is composed of diagonal elements and checking the multiset correspondence of indices between D and $\bar{D}$ factors. This specialized engine avoids Weingarten functions entirely, providing significant performance gains for phase-averaging tasks.
- **Memoization**: All computed characters, dimensions, and Weingarten values are cached using the `Memoization.jl` package. Memoization occurs at the granular level of individual character and Weingarten function call, ensuring maximum reuse across different partition structures and integration orders.

5.3 Symbolic trace logic

To handle high-level expressions, `IntegrateUnitary.jl` introduces specialized abstract types: `LazyTrace` and `LazySum`. These structures maintain the algebraic form of trace products (e.g., $\mathrm{tr}(AB)\mathrm{tr}(C)$) without strictly evaluating them into scalar components.

The integration strategy for these structures is twofold:

(1) **Library Lookup**: Common patterns, such as the single-channel overlap $\mathrm{tr}(UAU^\dagger B)$, are detected via pattern matching mechanisms (implemented in `check_library`). These matches trigger optimized fast-paths that return pre-computed results directly (e.g., $\frac{1}{d}\mathrm{tr}(A)\mathrm{tr}(B)$) without invoking the full Weingarten engine. This effectively reduces the complexity for these common cases from $O((k!)^2)$ to $O(1)$.
(2) **Graph Reconstruction**: For generic cases, the package interprets the trace cycles as edges in a graph. By identifying the relative positions of U and $U^\dagger$ within the lazy structure, the integrator assigns effective indices to the intervening constant scaling matrices. This effectively converts the "index-free" input into the precise tensor contraction graph required by the Weingarten formula calculation, thereby avoiding the combinatorial explosion associated with eager index expansion.

Input validation. `IntegrateUnitary.jl` validates user inputs at the API boundary to provide clear error messages. The package checks that Stiefel manifold parameters satisfy $k \leq d$, that partial trace subsystem indices are within range, and that t-design integrals do not exceed the design order t. For the symplectic group, dimensions must be even. For Haar-unitary rules (`dU`, and the current `dSU` backend), unbalanced monomials are symmetry-forbidden and evaluate to zero. `ArgumentError` is reserved for invalid measure constraints (e.g., odd symplectic dimension, or a t-design query beyond the design order).

6 Performance and benchmarks

We evaluated the performance of `IntegrateUnitary.jl` using a suite of stress tests involving high-degree polynomial integrals over the unitary, orthogonal, and symplectic groups. The benchmarks were run on a workstation equipped with an Intel Core i7-12700KF processor (12 cores, 20 threads) and 64 GB of RAM (Julia 1.12.3). Specifically, we evaluated high-degree entry moments, such as $\int |U_{11}|^{2k} dU$ for $2k \geq 8$, together with mixed-index and tensor-network workloads. The entry-moment cases test high-degree symbolic arithmetic and shortcut dispatch, while the mixed-index and tensor-network cases exercise the general combinatorial contraction paths.

All native `IntegrateUnitary.jl` benchmarks in this section were performed using the `BenchmarkTools.jl` package, and reported execution times represent the median of $N = 30$ cold-cache samples, with memoization caches cleared before each sample. These results *exclude* the initial JIT compilation and warm-up time, reflecting steady-state performance observed during production use. Cross-tool comparisons (Sections 6.4 and 6.5) use dedicated benchmark scripts with adaptive sample counts for very slow cases; those policies are stated in the corresponding subsections. The reproduction script `benchmarks/00_manuscript_benchmarks.jl` generates the native `IntegrateUnitary.jl` benchmark tables in this section and can be executed via the runner described in Section 8. Cross-tool comparisons (Haarpy/RTNI) are generated by dedicated scripts in `benchmarks/haarpy_benchmarks/` and `benchmarks/rtni_benchmarks/`. Additional stress tests and correctness checks are available in `benchmarks/09_stress_test_2.jl` and `benchmarks/08_stress_test_1.jl`.

6.1 Theoretical complexity

The computational complexity of generic Weingarten integration is governed by summation over the symmetric group S_k (for the unitary case) or the set of pair partitions M_{2k} (for orthogonal/symplectic cases). The size of these sets grows factorially: $|S_k| = k!$ and $|M_{2k}| = (2k-1)!!$. Consequently, the uncompressed formulas scale exponentially with the degree of the polynomial.

For the unitary group, the generic uncompressed formula contains a double sum over S_k, giving $O((k!)^2)$ terms per integral. The implementation groups terms by cycle type where useful, and symmetric entry moments such as $|U_{11}|^{2k}$ use closed-form or row/column moment shortcuts. Individual Weingarten function evaluations require summing over partitions $\lambda \vdash k$, each involving a character computation via the Murnaghan-Nakayama rule in $O(k^2)$ time per partition, and a Hook-Content dimension evaluation in $O(k)$ time. These values are memoized, so repeated queries are $O(1)$.

For the orthogonal and symplectic groups, the naive Gram matrix is indexed by pair partitions and has $(2k-1)!! \times (2k-1)!!$ entries. The implementation's main integration path instead solves a reduced system indexed by loop/cycle types and sums grouped pair-partition contractions. The reduced Weingarten data are cached once per degree and dimension, and subsequent queries reuse the cached values. Symplectic integrals use the same reduced data with $d \to -d$, together with the symplectic contraction signs, so they do not require a separate full matrix inversion.

Empirically, we find that exact symbolic integration is feasible for degrees up to $2k \approx$ 10-12 on standard hardware. Beyond this regime, the number of terms (e.g., $10! \approx 3.6 \times 10^6$) becomes prohibitive for real-time symbolic manipulation, although numerical evaluation remains tractable for slightly higher degrees.

Memory scaling. Memory consumption is governed by two factors: the size of the intermediate symbolic expressions and the memoization cache. For the unitary group, the number of partitions $\lambda \vdash k$ grows sub-exponentially (the partition function $p(k)$), and each cached Weingarten value is a single rational function of d, so the cache remains modest (under 1 MiB for $k \leq 6$). For the orthogonal and symplectic groups, the uncompressed pair-partition matrix would have $(2k-1)!! \times (2k-1)!!$ entries, but the main implementation stores and reuses reduced loop/cycle-type data. The dominant memory cost of generic intermediate symbolic expansions still grows rapidly with the number of valid contractions. Once memoized, however, subsequent integrations at the same degree k incur little additional memory, as Weingarten data are retrieved from cache rather than recomputed.

6.2 Benchmark results

Table 2 summarizes the execution times for selected integrals. Key observations include:

(1) **Symbolic overhead**: Computing integrals with symbolic dimension d is computationally demanding, but optimization strategies significantly reduce this cost. For example, computing $\int |U_{11}|^{10} dU$ symbolically took approximately 0.90 ms, achieving performance comparable to fixed $d = 10$ numeric integration (0.81 ms). This efficiency leverages the grouping of identical cycle structures in the Weingarten sum.

(2) **Low-degree performance**: For common use cases involving lower-degree polynomials (e.g., $2k = 6$), the package is extremely fast. The symbolic integration of $|U_{11}|^6$ completes in approximately 0.88 ms. For the orthogonal group, symbolic low-degree moments are similarly fast (O_{11}^2: 0.02 ms, O_{11}^4: 0.03 ms). For higher-degree orthogonal moments, the table reports concrete-dimension runs, where O_{11}^{10} remains sub-millisecond at $d = 20$ and $d = 50$ (0.37 ms and 0.36 ms).

(3) **Group differences**: Performance is workload-dependent. Unitary symbolic entry moments remain near the 1 ms range in this table, and orthogonal moments are highly efficient (especially in concrete high-degree cases). In contrast, the listed symplectic entry moments are noticeably slower at higher degree (4.51 ms for $|Sp_{11}|^8$ at $d = 10$, and $\sim$ 78–79 ms for $|Sp_{11}|^{10}$ at $d = 10, 20$). The dedicated comparison tables in Sections 6.4 and 6.5 show that the largest cross-tool speedups occur for high-degree unitary and orthogonal workloads, while some low-degree mixed-index unitary cases are near parity.

(4) **Correctness**: All benchmarks were verified against known exact values or asymptotic results, providing correctness checks for the reported cases.
(5) **Application**: The integration of the trace of the squared reduced density matrix for a bipartite state ($d = 6$), which involves contracting a 4^{th} degree polynomial over unitary group, completes in approximately 9.81 ms, demonstrating the tool's applicability to quantum information tasks.
(6) **Tensor network scaling**: The graphical engine scales efficiently with both the degree k and dimension d. As shown in Table 3, for a loop network of Haar unitaries, the median time increases from 0.02 ms at $k = 1$ to 15.44 ms at $k = 4$ for fixed $d = 2$. Dimension scaling for $k = 2$ remains sub-second up to $d = 50$, reaching 1.08 s at $d = 100$. For higher degrees like $k = 3$, the overhead increases more rapidly with d, with $d = 30$ taking 17.05 s, reflecting the increasing rank of intermediate tensor contractions.
(7) **Permutation groups**: Integration over the symmetric group S_d is exceptionally fast due to its combinatorial nature. Computing the degree-10 monomial $\prod_{i=1}^{10} P_{ii}$ (ten distinct index pairs) for $d = 100$ takes only 0.49 ms. This high performance extends to the centered permutation ensemble, which remains efficient even after polynomial expansion.

Table 2. Benchmark results for symbolic vs. numeric integration. Times are median values over 30 samples.

Group	Integrand	Dimension d	Time (ms)
Unitary	$\lvert U_{11}\rvert^6$	Symbolic	0.88
Unitary	$\lvert U_{11}\rvert^8$	Symbolic	0.90
Unitary	$\lvert U_{11}\rvert^{10}$	Symbolic	0.90
Unitary	$\lvert U_{11}\rvert^{10}$	$d = 10$	0.81
Unitary	$\lvert U_{11}\rvert^{10}$	$d = 50$	0.83
Orthogonal	O_{11}^2	Symbolic	0.02
Orthogonal	O_{11}^4	Symbolic	0.03
Orthogonal	O_{11}^6	$d = 10$	0.37
Orthogonal	O_{11}^8	$d = 20$	0.38
Orthogonal	O_{11}^{10}	$d = 20$	0.37
Orthogonal	O_{11}^{10}	$d = 50$	0.36
Symplectic	$\lvert Sp_{11}\rvert^8$	$d = 10$	4.51
Symplectic	$\lvert Sp_{11}\rvert^{10}$	$d = 10$	77.53
Symplectic	$\lvert Sp_{11}\rvert^{10}$	$d = 20$	79.18
GinUE	$\mathrm{tr}(GG^\dagger)$	Symbolic	0.00
GinUE	$\mathrm{tr}(GG^\dagger)$	$d = 4$	0.00
Circ. Orthogonal	$\lvert S_{11}\rvert^2$	Symbolic	0.02
Circ. Orthogonal	$\lvert S_{11}\rvert^4$	Symbolic	0.03
Circ. Orthogonal	$\lvert S_{11}\rvert^6$	Symbolic	1.07
Circ. Symplectic	$\lvert S_{11}\rvert^2$	Symbolic	0.30
Circ. Symplectic	$\lvert S_{11}\rvert^4$	Symbolic	2.92
Circ. Symplectic	$\lvert S_{11}\rvert^6$	Symbolic	102.43
Permutation	$\prod_{i=1}^{10} P_{ii}$	$d = 100$	0.49
Permutation	$\mathrm{tr}(PA)^2$	$d = 4$	14.57
Centered Perm.	Y_{11}^4	$d = 10$	0.43
Application	Bipartite	$d = 6$	9.81

The results demonstrate that `IntegrateUnitary.jl` remains efficient for practically relevant problem sizes. The rapid execution of the bipartite state application ($\approx$ 9.81 ms) highlights the library's utility for routine quantum

information tasks, while the sub-millisecond performance for low-degree orthogonal integrals facilitates large-scale statistical sampling. We further note that circular ensemble timings are heterogeneous: COE rows remain near 0.02–1.07 ms in Table 2, while higher-degree CSE examples are substantially more expensive. For highly symmetric integrands, such as powers of a single matrix entry, the integration engine utilizes an optimized grouped-summation technique that significantly reduces symbolic overhead by collecting terms with identical cycle types. Consequently, high-degree moments such as $|O_{11}|^6$ are computed in under 0.40 ms. Simultaneously, the symbolic engine proves capable of handling high-degree moments that are intractable by hand. The caching mechanism ensures that subsequent calls with the same parameters are virtually instantaneous.

Table 3. Benchmark results for ITensor network integration (Haar measure) showing scaling with degree k and dimension d. Timings exclude construction of the random ITensor constants and index objects.

Scaling type	Degree k	Dimension d	Time (ms)
Degree k (U)	1	2	0.01
Degree k (U)	2	2	0.05
Degree k (U)	3	2	0.42
Degree k (U)	4	2	13.02
Degree k (O)	2	3	0.01
Degree k (O)	4	3	0.33
Degree k (O)	6	3	463.25
Dimension d ($k = 2$)	2	2	0.05
Dimension d ($k = 2$)	2	10	0.22
Dimension d ($k = 2$)	2	50	73.42
Dimension d ($k = 2$)	2	100	1009.99
Dimension d ($k = 3$)	3	2	0.41
Dimension d ($k = 3$)	3	10	14.49
Dimension d ($k = 3$)	3	20	1470.44
Dimension d ($k = 3$)	3	30	16141.72
Orthogonal	6	3	416.86

6.3 Matrix integration benchmarks

We evaluated the performance of generic matrix integration by computing $\mathbb{E}[UU^\dagger]$ for $N \times N$ symbolic matrices over $U(d)$. The results (Table 4) demonstrate efficient scaling, as IntegrateUnitary automatically handles the element-wise operations.

Table 4. Benchmark results for matrix integration $\mathbb{E}[UU^\dagger]$ over $U(d)$. Timings are for constructing and integrating the full $N \times N$ matrix of expressions.

Matrix size (N)	Median time (ms)	Allocations (MiB)
2×2	2.55	0.82
3×3	6.58	2.22
4×4	12.53	4.84

6.4 Performance comparison with Haarpy

To provide a direct quantitative comparison with existing tools, we benchmarked `IntegrateUnitary.jl` against `Haarpy` [Cardin et al. 2024], the most closely comparable open-source package. Both packages were evaluated on the same machine using identical integrands. We tested both *diagonal* moments ($|M_{11}|^{2k}$), which admit closed-form shortcuts, and *off-diagonal* products (e.g., $|U_{11}|^2|U_{12}|^2$), which exercise the general Weingarten summation path. On the `IntegrateUnitary.jl` side, each row reports the median over fixed $N = 30$ cold-cache samples, with memoization caches cleared before every sample. On the `Haarpy` side, each row reports the median with default $N = 30$ samples. The script includes probe-based adaptive reduction to $N = 5$ for very slow cases, but this rerun stayed at $N = 30$ for all reported rows. Timings exclude JIT warm-up for `IntegrateUnitary.jl` and import overhead for `Haarpy`. Speedups are computed from these per-row medians. The comparison scripts are available in `benchmarks/haarpy_benchmarks/`.

Table 5 summarizes the results. For unitary integrals, `IntegrateUnitary.jl` achieves strong speedups for diagonal moments ($|U_{11}|^{2k}$, $k = 3, 4, 5$): 12.0×, 27.5×, and 88.5× in the symbolic-d rows, and 47.5×-55.3× in the numeric-d rows. Off-diagonal unitary integrals range from near parity (1.5× for $|U_{11}|^2|U_{22}|^2$) to substantial gains (11.4× for $|U_{11}|^4|U_{12}|^4$).

The orthogonal group results reveal a more dramatic separation: while both packages handle low-degree cases efficiently, `IntegrateUnitary.jl` is already substantially faster (O^2_{11}: 34.7×, O^4_{11}: 70.0×), `IntegrateUnitary.jl`'s optimized univariate Weingarten solver delivers speedups exceeding 2.7×10^4× at degree 10. This disparity arises because `Haarpy` constructs and inverts the full $(2k-1)!! \times (2k-1)!!$ Weingarten matrix even when a single entry suffices, whereas `IntegrateUnitary.jl` exploits the structure of univariate integrands to bypass this cost entirely. Off-diagonal orthogonal integrals show a 4.5-6.0× advantage.

For the circular orthogonal ensemble (COE), `IntegrateUnitary.jl` uses a closed-form expression for diagonal moments, yielding 20.9-127.2× speedups that grow with the degree. We note that `Haarpy` (v0.0.6, the latest release available at time of writing) returned incorrect results (zero) for the off-diagonal COE integrals tested ($|S_{12}|^{2k}$, $|S_{11}|^2|S_{12}|^2$), which `IntegrateUnitary.jl` evaluates correctly; these cases are therefore excluded from the table.

Overall, `IntegrateUnitary.jl` demonstrates a substantial performance advantage across most tested groups, with the largest gains in high-order orthogonal and unitary diagonal moments.

6.5 Performance comparison with RTNI

We benchmark `IntegrateUnitary.jl` against `RTNI` [Fukuda et al. 2019] on identical integrands over $U(d)$, executed on the same hardware. Table 6 reports median runtimes (default $N = 10$ samples, adaptively reduced to $N = 3$ for slow cases; JIT/import overhead excluded). The speedup is defined as RTNI/IntegrateUnitary.jl, so values greater than one indicate faster execution in `IntegrateUnitary.jl`. All rows produce fully evaluated scalar results in both tools, ensuring an apples-to-apples comparison. For `RTNI`'s graph-based engine, this includes the cost of converting the internal graph representation to a scalar expression via `converttomonomial`. Reproduction scripts are available in `benchmarks/rtni_benchmarks/`.

For element-wise integrals, the performance gap grows dramatically with polynomial degree. At the lowest order ($|U_{11}|^2$), `RTNI` is moderately faster (0.3×). By degree eight ($|U_{11}|^8$), `IntegrateUnitary.jl` is over three orders of magnitude faster (3365×-4072×), reflecting `IntegrateUnitary.jl`'s optimized cycle-grouping strategy that avoids the full permutation-level expansion. For mixed-index integrals, performance is near parity at low degree (0.9×-1.0× for $|U_{11}|^2|U_{12}|^2$) and diverges at higher degree (14×-15× for $|U_{11}|^2|U_{12}|^4$). For trace-polynomial integrals, $\mathrm{tr}(UAU^*B)$ and $\mathrm{tr}((UAU^*B)^2)$, `IntegrateUnitary.jl` shows advantages of 2.8× and 1.9×, respectively.

Table 5. Performance comparison: `IntegrateUnitary.jl` vs. `Haarpy` (IntegrateUnitary.jl: fixed $N = 30$ cold-cache samples; Haarpy: default $N = 30$ with probe-based reduction to $N = 5$ enabled, not triggered in this run; speedup = Haarpy / IntegrateUnitary.jl from row medians). Both packages were run on the same hardware; timings exclude JIT/import overhead.

Group	Integrand	IntegrateUnitary.jl (ms)	Haarpy (ms)	Speedup
Diagonal moments				
U(d)	$\|U_{11}\|^6$, symbolic d	0.99	11.85	12.0×
U(d)	$\|U_{11}\|^8$, symbolic d	0.88	24.10	27.5×
U(d)	$\|U_{11}\|^{10}$, symbolic d	0.89	78.65	88.5×
U(d)	$\|U_{11}\|^{10}$, $d = 10$	1.00	47.38	47.5×
U(d)	$\|U_{11}\|^{10}$, $d = 50$	0.82	45.29	55.3×
O(d)	O_{11}^2, symbolic d	0.02	0.74	34.7×
O(d)	O_{11}^4, symbolic d	0.06	3.89	70.0×
O(d)	O_{11}^6, $d = 10$	0.36	0.91	2.5×
O(d)	O_{11}^8, $d = 20$	0.37	63.17	172.7×
O(d)	O_{11}^{10}, $d = 20$	0.35	9851.39	27942.2×
O(d)	O_{11}^{10}, $d = 50$	0.53	9756.51	18408.3×
COE	$\|S_{11}\|^2$, symbolic d	0.02	2.37	107.7×
COE	$\|S_{11}\|^4$, symbolic d	0.04	4.97	127.2×
COE	$\|S_{11}\|^6$, symbolic d	0.95	19.85	20.9×
Off-diagonal products				
U(d)	$\|U_{11}\|^2\|U_{12}\|^2$, symbolic d	1.50	6.42	4.3×
U(d)	$\|U_{11}\|^4\|U_{12}\|^4$, symbolic d	1.90	21.71	11.4×
U(d)	$\|U_{11}\|^2\|U_{22}\|^2$, symbolic d	2.06	3.10	1.5×
O(d)	$O_{11}^2 O_{12}^2$, symbolic d	0.60	3.61	6.0×
O(d)	$O_{11}O_{12}O_{21}O_{22}$, sym. d	1.09	4.90	4.5×

Table 6. Performance comparison: `IntegrateUnitary.jl` vs. `RTNI` (Mathematica). Median runtime over repeated runs. All rows produce fully evaluated scalar results in both tools.

Group	Integrand	IntegrateUnitary.jl (ms)	RTNI (ms)	Speedup
U(d) (Element API)	$\|U_{11}\|^2$, symbolic d	1.34	0.53	0.4×
U(d) (Element API)	$\|U_{11}\|^4$, symbolic d	1.49	2.51	1.7×
U(d) (Element API)	$\|U_{11}\|^6$, symbolic d	1.42	37.81	26.5×
U(d) (Element API)	$\|U_{11}\|^8$, symbolic d	1.41	5275.56	3745.2×
U(d) (Element API)	$\|U_{11}\|^8$, $d = 10$	1.33	5123.40	3852.1×
U(d) (Element API)	$\|U_{11}\|^2\|U_{12}\|^2$, symbolic d	2.26	2.49	1.1×
U(d) (Element API)	$\|U_{11}\|^2\|U_{12}\|^4$, symbolic d	2.42	36.29	15.0×
U(d) (Element API)	$\|U_{11}\|^2\|U_{22}\|^2$, symbolic d	2.85	2.48	0.9×
U(d) (Element API)	$\|U_{11}\|^2\|U_{12}\|^2$, $d = 10$	2.32	2.46	1.1×
U(d) (Element API)	$\|U_{11}\|^2\|U_{12}\|^4$, $d = 10$	2.27	35.42	15.6×
U(d) (Element API)	$\|U_{11}\|^2\|U_{22}\|^2$, $d = 10$	2.47	2.43	1.0×
U(d)	$\|\mathrm{tr}(U)\|^4$, $d = 10$	0.01	0.18	19.9×
U(d)	$\|\mathrm{tr}(U)\|^6$, $d = 10$	0.01	0.19	16.9×
U(d)	$\|\mathrm{tr}(U)\|^8$, $d = 10$	0.02	0.21	12.0×
U(d)	$\mathrm{tr}(UAU^*B)$, symbolic d	0.13	0.50	3.9×
U(d)	$\mathrm{tr}((UAU^*B)^2)$, symbolic d	1.37	2.45	1.8×

7 Conclusion

We have presented `IntegrateUnitary.jl`, a robust and extensible Julia package for performing symbolic integration over the Haar measure of the unitary, orthogonal, and symplectic groups (and $SU(d)$ for balanced polynomials). By bridging the gap between the abstract theory of Weingarten calculus and practical computation, `IntegrateUnitary.jl` empowers researchers to rigorously derive properties of random quantum channels, higher-order entanglement statistics, and scrambling measures without resorting to brittle or approximate numerical sampling.

The package's core strengths lie in its unified treatment of compact groups, its broad symbolic-d support for entry-wise and trace-polynomial integrals (with explicit concrete-d exceptions for higher pure trace moments, HCIZ on `SymbolicMatrix` inputs, and direct matrix-valued integration of `SymbolicMatrix`/`SymbolicMatrixProduct` expressions), and its novel symbolic trace logic. The latter effectively decouples the user's high-level mathematical intent from the low-level index contractions required by the integration engine, significantly reducing the cognitive load and potential for error. As demonstrated by the benchmarks, the optimized Julia implementation ensures that this symbolic abstraction does not come at the cost of performance, enabling the evaluation of high-degree moments that were previously intractable.

Future development of `IntegrateUnitary.jl` will focus on expanding support to exceptional Lie groups (particularly G_2, which arises in holonomy classification and string-theoretic compactifications), integrating finite-d corrections for $SU(d)$ beyond the stable range via ϵ-tensor contractions, and further optimizing the asymptotic expansion engine for extremely large-scale problems. We believe `IntegrateUnitary.jl` will become an essential utility in the toolkit of theoretical physicists and quantum information scientists.

8 Software availability and reproducibility

The source code for `IntegrateUnitary.jl` is available on GitHub under the Apache License 2.0.

- **Repository URL**: https://github.com/iitis/IntegrateUnitary.jl
- **Archived release DOI**: https://doi.org/10.5281/zenodo.20346848
- **License**: Apache License 2.0
- **Version**: The results in this paper were generated using version v1.0.0.
- **Pinned Julia environments**: Reproducibility relies on versioned `Manifest.toml` files in `examples/` and `benchmarks/`. The benchmark/example runner scripts instantiate these environments before execution.
- **Benchmarks**: Native IntegrateUnitary Julia benchmarks are turnkey from a clean checkout (with pinned Julia manifests). To reproduce them, run:

```
bash benchmarks/runbenchmarks.sh
```

Cross-tool comparisons (`Haarpy`, `RTNI`) require manual external setup and are not one-command reproducible from Julia manifests alone.
Haarpy setup and run:

```
cd benchmarks/haarpy_benchmarks
conda env create -f environment.haarpy_bench.yml
```

```
    conda activate haarpy_bench
    bash run_benchmarks.sh
```

The pinned conda environment specifies `haarpy==0.0.6` (overrideable at runtime via `HAARPY_VERSION`). RTNI setup and run:

```
    # 1) Install RTNI for Mathematica and stage RTNI.wl + precomputedWG/
    #    in benchmarks/rtni_benchmarks/ (or on $Path)
    cd benchmarks/rtni_benchmarks
    export
        RTNI_EXPECTED_SHA256=cf7aaa1ba49b249ce7b9dd4a682bf2dd4cf83b47b8b92b341a84a882238694b2
    bash run_benchmarks.sh
```

For the manuscript comparison, the RTNI source revision is pinned by `RTNI_EXPECTED_SHA256` (full value shown in the command block above). The RTNI workflow records the package identifier in metadata (version symbol when available, otherwise `RTNI.wl` SHA-256). Each generated comparison result file includes an optional `_meta` block (runtime/package versions, timestamp, host/OS details, and source identifiers) to document the exact run context across machines.

- **Manuscript build**: Build the paper from the `txt/` directory so relative figure/table paths resolve correctly:

```
    cd txt
    latexmk -pdf manuscript.tex
```

Acknowledgments

ZP and ŁP acknowledge support from the National Science Center (NCN), Poland, under Project Opus No. 2022/47/B/ST6/02380.

References

Andris Ambainis and Joseph Emerson. 2007. Quantum t-designs: t-wise independence in the quantum world. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*. IEEE, 129–140.

Erwin H. Bareiss. 1968. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comp.* 22, 103 (1968), 565–578. https://doi.org/10.1090/S0025-5718-1968-0226829-0

Ingemar Bengtsson and Karol Życzkowski. 2017. *Geometry of quantum states: an introduction to quantum entanglement* (2 ed.). Cambridge University Press.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM review* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671

Fernando GSL Brandao, Aram W Harrow, and Michał Horodecki. 2016. Local random quantum circuits are approximate polynomial-designs. *Communications in Mathematical Physics* 346, 2 (2016), 397–434. https://doi.org/10.1007/s00220-016-2706-8

Yanic Cardin, Hubert de Guise, and Nicolás Quesada. 2024. Haarpy, a Python library for Weingarten calculus and integration of classical compact groups and ensembles. https://github.com/polyquantique/haarpy.

Benoît Collins. 2003. Moments and cumulants of polynomial random variables on unitary groups, the Itzykson-Zuber integral, and free probability. *International Mathematics Research Notices* 2003, 17 (2003), 953–982. https://doi.org/10.1155/S107379280320917X

Benoît Collins and Piotr Śniady. 2006. Integration with respect to the Haar measure on unitary, orthogonal and symplectic groups. *Communications in Mathematical Physics* 264, 3 (2006), 773–795. https://doi.org/10.1007/s00220-006-1554-3

Christoph Dankert, Richard Cleve, Joseph Emerson, and Etera Livine. 2009. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Physical Review A* 80, 1 (2009), 012304. https://doi.org/10.1103/PhysRevA.80.012304

Alan Edelman, Tomás A Arias, and Steven T Smith. 1998. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications* 20, 2 (1998), 303–353. https://doi.org/10.1137/S089547989833276X

Matthew Fishman, Steven R. White, and E. Miles Stoudenmire. 2022. The ITensor Software Library for Tensor Network Calculations. *SciPost Phys. Codebases* (2022), 4. https://doi.org/10.21468/SciPostPhysCodeb.4

Peter J Forrester. 2010. *Log-Gases and Random Matrices (LMS-34).* Princeton University Press.

Motohisa Fukuda, Robert König, and Ion Nechita. 2019. RTNI—A symbolic integrator for Haar-random tensor networks. *Journal of Physics A: Mathematical and Theoretical* 52, 42 (2019), 425303. https://doi.org/10.1088/1751-8121/ab434b

Jean Ginibre. 1965. Statistical ensembles of complex, quaternion, and real matrices. *J. Math. Phys.* 6, 3 (1965), 440–449. https://doi.org/10.1063/1.1704292

Shashi Gowda, Yingbo Ma, Alessandro Cheli, Maja Gwóźdź, Viral B. Shah, Alan Edelman, and Christopher Rackauckas. 2022. High-Performance Symbolic-Numerics via Multiple Dispatch. *ACM Communications in Computer Algebra* 55, 3 (2022), 92–96. https://doi.org/10.1145/3511528.3511535

David Gross, Koenraad Audenaert, and Jens Eisert. 2007. Evenly distributed unitaries: On the structure of unitary designs. *J. Math. Phys.* 48, 5 (2007), 052104. https://doi.org/10.1063/1.2716992

Harish-Chandra. 1958. Spherical Functions on a Semisimple Lie Group, I. *American Journal of Mathematics* 80, 2 (1958), 241–310. https://doi.org/10.2307/2372786

Claude Itzykson and Jean-Bernard Zuber. 1980. The planar approximation. II. *J. Math. Phys.* 21, 3 (1980), 411–421. https://doi.org/10.1063/1.524438

Giacomo Livan, Marcel Novaes, and Pierpaolo Vivo. 2018. *Introduction to Random Matrices: Theory and Practice.* SpringerBriefs in Mathematical Physics, Vol. 26. Springer. https://doi.org/10.1007/978-3-319-70885-0

Madan Lal Mehta. 2004. *Random matrices.* Elsevier.

Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information.* Cambridge University Press.

Don N Page. 1993. Average entropy of a subsystem. *Physical Review Letters* 71, 9 (1993), 1291–1294. https://doi.org/10.1103/PhysRevLett.71.1291

Zbigniew Puchała and Jarosław Adam Miszczak. 2017. Symbolic integration with respect to the Haar measure on the unitary group. *Bulletin of the Polish Academy of Sciences: Technical Sciences* 65, 1 (2017), 21–27. https://doi.org/10.1515/bpasts-2017-0003

John Watrous. 2018. *The theory of quantum information.* Cambridge University Press.

Don Weingarten. 1978. Asymptotic behavior of group integrals in the limit of infinite rank. *J. Math. Phys.* 19, 5 (1978), 999–1001. https://doi.org/10.1063/1.523807

Gian Carlo Wick. 1950. The evaluation of the collision matrix. *Physical Review* 80, 2 (1950), 268–272. https://doi.org/10.1103/PhysRev.80.268

Karol Życzkowski and Hans-Jürgen Sommers. 2001. Induced measures in the space of mixed quantum states. *Journal of Physics A: Mathematical and General* 34, 35 (2001), 7111–7125. https://doi.org/10.1088/0305-4470/34/35/335