Learning *Quantum Integrable* Structure with Artificial Intelligence: A Case of AI-Led Scientific Research

Justinian K. Wang[1,2]

[1] *Yau Mathematical Sciences Center, Tsinghua University, Beijing 100084, China*
[2] *School of Physics, Nanjing University, Nanjing 210023, China*

(Dated: November 9, 2025)

Modern artificial intelligence (AI) systems have demonstrated remarkable potential in exploring foundational problems in physics. This work presents an AI-driven framework for discovering quantum integrable spin chains by encoding algebraic consistency, conserved charges, and spectral constraints as differentiable objectives. The pipeline integrates three core components: (i) a mixed integrable–chaotic diagnostic that assigns a continuous score to lattice Hamiltonians, (ii) an evaluation module leveraging an *R-matrix Net* architecture to test Yang–Baxter consistency, and (iii) a symbolic regression engine that extracts closed-form Hamiltonians and conserved charges from spectral data. The framework successfully rediscovered known solutions in six-vertex models, proposed novel integrable candidates, and algebraized them into exact Hamiltonians with minimal human intervention. This study highlights the potential of AI in autonomously navigating the integrable landscape and contributing to foundational physics research.

CONTENTS

INTRODUCTION

Quantum integrability offers a rigorous testbed for artificial intelligence. Local spin chains, characterized by commuting charges derived from R-matrices satisfying the Yang–Baxter equation (YBE), provide a structured framework for exploring integrable systems. Recent advancements have demonstrated that these constraints can be reformulated as learning objectives. For instance, the *R-matrix Net* architecture incorporates YBE residuals, regularity, and hermiticity as loss terms, enabling the discovery of both known and novel integrable Hamiltonians [1].

AI has also been employed to uncover conserved quantities, symmetries, and closed-form laws directly from data. In dynamical systems, neural networks have identified conservation laws and hidden symmetries [2, 3], while symbolic regression tools like AI Feynman 2.0 have distilled compact analytic expressions from high-accuracy fits [4]. In high-energy physics and algebraic geometry, machine learning has been applied to classify large datasets and suggest new mathematical structures [5, 6].

This work builds on these developments by implementing a fully differentiable, constraint-driven pipeline for quantum integrability. The framework autonomously learns R-matrices and transfer matrices, explores nearby integrable candidates, and algebraizes these candidates into exact Hamiltonian families. This study serves as a case study in AI-led research, quantifying the extent to which modern systems can autonomously navigate the integrable landscape and produce proof-ready models.

This work presents a concrete case study of *AI-led scientific discovery* in quantum integrability. Rather than treating neural networks or symbolic tools as black-box auxiliaries, we design an end-to-end pipeline in which modern AI systems not only *analyze* data, but also *generate, operate, and refine* the very code artefacts that implement the physics logic. The result is a short, fully differentiable workflow that turns Yang–Baxter consistency, transfer-matrix algebra, and spectral/operational diagnostics into actionable loss functions and numerical tests, eventually feeding them into a symbolic back-end that outputs human-readable Hamiltonians and conserved charges.

At the implementation level, the framework is organized explicitly around three mutually reinforcing code components:

1. `final.py`, which hosts the *IntegrabilityDetector*, a no-diagonalization diagnostic that maps any $H \in \mathbb{C}^{n\times n}$ to a feature vector and a calibrated scalar *IntegrabilityScore* via four channels (local Reshetikhin residuals, Krylov/Lanczos growth, KPM spectral form factor, and sparse near-conserved operators).

2. `r_matrix_net_new.py`, which implements a compact *R-matrix Net* in the spirit of [7, 8]: it parameterizes the nonzero entries of a candidate $R(u)$ under a chosen sparsity pattern, trains on Yang–Baxter plus regularity losses, and differentiates at u=0 to produce local densities $h_{j,j+1}$ that are passed directly to the detector.

3. `pysr_4_inte_new.py`, which realizes a *symbolic-regression front-end* based on PySR and related ideas from AI Feynman [9], starting from high-precision data for $y(u) = \partial_u \log \Lambda_0(u)$ on XXZ-type transfer matrices and distilling compact "letter" expansions in coth-like building blocks that can be algebraized and checked.

These three scripts form a closed loop: `r_matrix_net_new.py` proposes and refines candidate integrable structures at the R-matrix level, `final.py` evaluates their integrability via operator and spectral diagnostics without ever performing exact diagonalization, and `pysr_4_inte_new.py` converts the resulting high-precision traces into exact formulas and algebraic varieties. Importantly, a substantial fraction of this code—from class skeletons to numerical routines—was produced and iteratively refined with large language models and code assistants, so that the pipeline itself is already a product of AI-guided programming. In this sense, the experiments reported here are not only about discovering integrable models *with* AI, but about letting AI design, maintain, and extend the computational infrastructure for integrable discovery.

I. PRELIMINARIES: ALGEBRAIC INTEGRABILITY

We retain only what the code uses. (i) RTT ⇒ commuting transfer matrices. With monodromy $T_a(u) = R_{aL}(u)\cdots R_{a1}(u)$ and $t(u) = \mathrm{Tr}_a\, T_a(u)$, the YBE implies the RTT relation $R_{ab}(u-v)T_a(u)T_b(v) = T_b(v)T_a(u)R_{ab}(u-v)$ and hence $[t(u), t(v)] = 0$ for all u, v; our tests evaluate this numerically on a grid. (ii) Charges and H from small-u. Under regularity $R(0) = P$, $t(0)$ is lattice translation and $\log t(u) = \sum_{n\geq 0} \frac{u^n}{n!} Q_{n+1}$ with $[Q_m, Q_n] = 0$; the two-site density is $h_{j,j+1} \propto \partial_u \check{R}_{j,j+1}(u)\,|_{u=0}$ with $\check{R}(u) = PR(u)$, implemented by automatic differentiation. (iii) Boost recursion as a local witness. With $B = \sum_j j\, h_{j,j+1}$ one has $Q_{n+1} \propto [B, Q_n]$, in particular $Q_3 \propto \sum_j [h_{j,j+1}, h_{j+1,j+2}]$; checking $[Q_2, Q_3] \approx 0$ is our fast local integrability screen before full RTT. (iv) Reshetikhin/CYBE consistency for algebraization. Writing $\check{R}(u) = \mathbb{I} + uH + \frac{u^2}{2}K + \cdots$, the braided YBE enforces $\mathcal{O}(uv)/\mathcal{O}(u^2, v^2)$ constraints used only when lifting neural $R(u)$ to closed forms (PSLQ/Prony). (v) Gauge handling. $(G\otimes G)R(f(u))(G^{-1}\otimes G^{-1})$ and scalar reweightings preserve YBE, so we normalize by simple gauges without affecting physics. Canonical references: [10–15].

Implementation notes (algorithm-friendly forms used in `final.py`).

1. *Minimal input:* a set of nearest-neighbor densities h (or a parameterized ansatz $h(\theta)$). Numerically one first removes any identity component, aligning with the scalar freedom described in the gauge-normalization discussion.
2. *Fast necessary test:* construct Q_3 via the explicit local commutator formula and compute the operator norm / residual of $[Q_2, Q_3]$ as a diagnostic ($\to$ the *cons residual* metric).
3. *Boost recursion:* generate approximate higher charges $Q_4, Q_5, \ldots$ using the standard relation $Q_{n+1} \propto [B, Q_n]$, and apply sparse linear regression / convex combination techniques to reduce ill-conditioning (aligned with SINDy ideas).
4. *Cross-validation with the commuting family and spectral statistics:* if the previous steps pass, construct a truncated expansion of $t(u)$ for small system sizes and verify $[Q_m, Q_n] \approx 0$; concurrently evaluate spectral form factor (SFF) ramps and level statistics to rule out manifestly chaotic behavior.
5. *(Optional) Reconstruction of R from h:* use Baxterization or initialize $\check{R}(u) = \mathbb{I} + uH + \cdots$ and minimize a YBE residual to build $R(u)$, enforcing regularity/unitarity/difference-form constraints, and finally apply symbolic regression to extract closed forms.

The above pipeline turns the textbook chain "YBE $\Rightarrow$ RTT $\Rightarrow$ $[t(u), t(v)] = 0 \Rightarrow \{Q_n\}$" into an *actionable* test/construct protocol: in code we prioritize computing $[Q_2, Q_3]$ (the cheapest and most discriminating check); passing this gate leads to higher-order and R-matrix level validations, while failure flags a "non-integrable-like" sample.

II. `INTEGRABILITY SCORER`: A MIXED DISTINGUISHER OF INTEGRABLE & CHAOTIC SYSTEM

a. Design goals. `final.py` implements an integrated "learn–diagnose" pipeline that does not require exact diagonalization. The core objects are the `IntegrabilityDetector` and the (optional) `RMatrixExplorer`. The former produces four-channel (A–D) features for arbitrary dense/sparse $H \in \mathbb{C}^{n\times n}$ and fuses them into a single scalar *IntegrabilityScore*; the latter parameterizes $R(u)$ by learnable nonzero entries, minimizes YBE/regularity losses, and differentiates at u=0 to obtain local densities $h_{j,j+1}$, which are then passed to the detector. This section aligns algorithms, code, and formulas and provides reproducible implementation notes.

Core API and Data Flow

b. Minimal API.

- `IntegrabilityDetector(H, L, seeds, cfg)`: construct the detector (only requiring that H supports `@` multiplication).
- `eval(H, hlist=None)`: run the four channels (A–D) and return a feature dictionary $\{\alpha, R^2_\alpha, \text{ramp}, R^2_{\text{ramp}}, \text{res}_{23}, \text{nnz}\}$.
- `score(features)`: normalize features and calibrate via a lightweight logistic regression, returning the *IntegrabilityScore* and a label.
- `RMatrixExplorer(sparsity)`: define a differentiable parameterization and loss for $R(u)$ given a sparsity pattern (e.g., six-vertex).

c. Data flow. (1) Optional: `RMatrixExplorer` learns $R(u)$ and produces $h_{j,j+1}$ at u=0; (2) `eval` computes the statistics for channels A–D; (3) `score` fuses the features and compares them to the synthetic calibrator (GOE/Poisson) to produce the final score.

A. Channel A (Algebraic): $[Q_2, Q_3]$ Fast Check

Inputs. Local density $h_{j,j+1}$ (skip if unavailable).
Construct. $Q_2 = \sum_j h_{j,j+1}$, $\quad Q_3 = \sum_j [h_{j,j+1}, h_{j+1,j+2}]$ (periodic boundary conditions).
Output. Reshetikhin residual

$$\text{res}_{23} = \frac{\|[Q_2, Q_3]\|_F}{\|Q_2\|_F \, \|Q_3\|_F}. \tag{1}$$

Rationale. If a regular braided $\check{R}(u)$ exists, then $H = \check{R}'(0)$ must satisfy the second-order expansion consistency; empirically $\text{res}_{23} \approx 0$ strongly indicates integrability (see the discussion around the local Q_3 construction in §I).
Pseudo-code.

1. Build a ring of size L and tile the local density h accordingly;
2. Sum using sparse block tridiagonal operations to form Q_2, compute local commutators to form Q_3;
3. Compute the Frobenius-normalized residual as the channel feature.

B. Channel B (Krylov): Lanczos Operator Growth

Inputs. A normalized seed O_0 (traceless diagonal or rank-one off-diagonal).
Recurrence. Under the Hilbert–Schmidt inner product $\langle A, B\rangle = \frac{1}{n}\mathrm{Tr}(A^\dagger B)$, perform a three-term Lanczos recurrence for the Liouvillian $\mathcal{L}(\cdot) = [H, \cdot]$:

$$\mathcal{L}O_k = b_{k+1}O_{k+1} + b_k O_{k-1}, \quad O_{-1} \equiv 0,\ b_k > 0. \tag{2}$$

Output. Fit the tail to $b_k \simeq \alpha k + \gamma$ and read off the slope α and fit quality R^2_α.
Complexity. The dominant cost is m Liouvillian applications plus orthogonalization: $\mathcal{O}(m \cdot T_{\mathrm{mv}})$, where T_{mv} denotes one matrix–vector product cost (near-linear for sparse/structured H).

C. Channel C (Spectral): KPM for SFF without Eigenpairs

Goal. Estimate $K(t) = |\mathrm{Tr}(e^{-itH})|^2$ without diagonalization and detect the dip–ramp–plateau structure.
Steps.

1. Linear rescaling: $H \mapsto H_s \in [-1, 1]$;
2. Chebyshev–Bessel expansion

$$e^{-itH_s} = J_0(t)I + 2\sum_{m\geq 1}(-i)^m J_m(t)\, T_m(H_s);$$

3. Use Hutchinson randomized trace estimation together with Jackson damping to estimate $\mathrm{Tr}(e^{-itH_s})$ at times t;
4. Choose adaptive time windows and polynomial order (guided by α), and fit the ramp to obtain slope and R^2_{ramp}.

Outputs. Ramp strength (`ramp`) and fit quality R^2_{ramp}, serving as a dynamical complement to Channel B.

D. Channel D (Symbolic/Sparse): Near-Conserved Charges

Goal. Find a sparse $X = \sum_i \theta_i A_i$ on a low-bandwidth dictionary $\{A_i\}$ such that $[H, X] \approx 0$.
Program.

$$\min_\theta \ \|[H, \sum_i \theta_i A_i]\|_F^2 + \lambda\|\theta\|_1. \tag{3}$$

Trick. Use randomized vectors z and the identity $\|[H, X]\|_F^2 = \mathbb{E}_z\|[H, X]z\|_2^2$ to construct a design matrix F without explicitly forming commutators. Solve with coordinate-descent / regularized least squares (LASSO), and return the residual and nnz (number of active coefficients).

E. Feature Fusion and Calibration

Normalization. Scale scalar features by conservative prior intervals (e.g. $\alpha \in [0, 1]$, $\mathrm{res}_{23} \in [0, 0.25]$).
Base score. Structural evidence (Channels A and D) is weighted more heavily than purely dynamical signals (Channels B and C).
Calibrator. Train a tiny logistic regressor on a minimal synthetic dataset (GOE for chaotic, sorted-diagonal for Poisson/integrable), obtain $p_{\mathrm{int}}(H)$, and average it with the base score to form the *IntegrabilityScore*, which is mapped to labels {`integrable-like`, `borderline`, `chaotic-like`}.

F. Complexity, Stability, and Defaults

Krylov (B). $\mathcal{O}(m \cdot T_{\mathrm{mv}})$ with m Lanczos steps and stable reorthogonalization.
KPM (C). Each Chebyshev moment $T_m(H_s)$ requires one `mv`; total cost is $\mathcal{O}(M \cdot P \cdot T_{\mathrm{mv}})$ where M is the Chebyshev order and P the number of probes. Jackson damping mitigates Gibbs ringing.
LASSO (D). The randomized design is formed from K trace probes; coordinate-descent complexity is near-linear in dictionary size and K.
Defaults. m=64, M=256, P=8, dictionary bandwidth w=2. If α is anomalously small, the pipeline adaptively increases M, the t-range, and w.

G. Reproducibility: Seeds, Caching, and CLI

Seeds. Explicitly set NumPy / PyTorch random seeds and fix probe sequences.
Caching. Cache calibrator weights and KPM windows keyed by matrix size n.
CLI. Provide flags such as `-mode {detect,explore}`, `-n`, `-krylov-steps`, `-kpm-order`, `-probes`, `-dict-bandwidth`, etc.

H. Limitations and Failure Modes

(i) Small-size ring validation in Channel A can yield finite-size false positives; (ii) near-integrable systems may lie near decision boundaries in Channels B and C; (iii) Channel D depends on dictionary choice and should be cross-validated with Channel A; (iv) non-difference-form / high-spin $R(u)$ require extending sparsity patterns and loss terms in the explorer.

I. Integrability Diagnostics Without Exact Diagonalization

This section provides a functional-level description of the code structure in `final v2.py`, with the aim that readers can directly map each subroutine to its corresponding physical or numerical object and thus reproduce experiments without performing exact diagonalization. The core class is `IntegrabilityDetector` (optionally paired with `RMatrixExplorer`); the detector requires only that $H \in \mathbb{C}^{n\times n}$ support matrix–vector products, while the explorer parameterizes $R(u)$ under a sparsity template and minimizes YBE and regularity losses, differentiates at $u = 0$ to obtain the local density $h_{j,j+1}$, and feeds it back to the detector. Internally the detector is organized into four "no-diagonalization" channels whose outputs are fused into a single scalar *IntegrabilityScore*. The data flow is: (i) if available, `RMatrixExplorer` first emits $h_{j,j+1}$; (ii) `eval` computes the four channel features in turn; (iii) `score` normalizes and lightly calibrates those features to return a score and a three-way label.

Channel A (algebraic fast checks) assembles on a short periodic chain

$$Q_2 = \sum_{j=1}^{L} h_{j,j+1}, \qquad Q_3 = \sum_{j=1}^{L} [h_{j,j+1},\, h_{j+1,j+2}],$$

and reports the Reshetikhin-type residual

$$\mathrm{res}_{23} = \frac{\|[Q_2, Q_3]\|_F}{\|Q_2\|_F\, \|Q_3\|_F}, \tag{4}$$

which serves as a rapid test for whether $[Q_2, Q_3] \approx 0$. This quantity is closely related to integrability classifications based on the YBE and regularity and is implemented as a strong criterion in nearest-neighbor and finite-range interaction settings; the code uses sparse block operations to avoid constructing large dense matrices explicitly. See recent systematic classifications and discussions of 4×4 / 8-vertex solutions [16, 17].

Channel B (Krylov–Lanczos operator growth) treats the Liouvillian $\mathcal{L}(\cdot) = [H, \cdot]$ and performs a three-term Lanczos recursion on a normalized seed O_0 under the Hilbert–Schmidt inner product $\langle A, B\rangle = \frac{1}{n}\,\mathrm{Tr}\big(A^\dagger B\big)$:

$$\mathcal{L}O_k = b_{k+1}O_{k+1} + b_k O_{k-1}, \qquad O_{-1} \equiv 0,\ b_k > 0, \tag{5}$$

recording $\{b_k\}$ and fitting the tail to $b_k \simeq \alpha k + \gamma$ to obtain the slope α and fit quality R^2_α. In chaotic systems α grows approximately linearly with k, while integrable or many-body-localized dynamics show significant deviations; hence α serves as a dynamic proxy. Implementation details include full reorthogonalization and adaptive truncation of the step count m [18**?**].

Channel C (KPM spectral form factor) estimates $K(t) = \big|\,\mathrm{Tr}\big(e^{-itH}\big)\big|^2$ without eigenpairs. The code linearly rescales $H \mapsto H_s \in [-1, 1]$ and employs the Bessel–Chebyshev expansion

$$e^{-itH_s} = J_0(t)\, I + 2\sum_{m\geq 1} (-i)^m J_m(t)\, T_m(H_s), \tag{6}$$

estimating Chebyshev moments with Hutchinson randomized trace probes

$$\mu_m = \mathrm{Tr}\big[T_m(H_s)\big] \approx \frac{1}{P}\sum_{p=1}^{P} (r^{(p)})^\dagger T_m(H_s)\, r^{(p)}, \tag{7}$$

and applying Jackson damping to reduce Gibbs oscillations. The time window and polynomial order are adapted using the Krylov slope α. The resulting $K(t)$ is scanned for the dip–ramp–plateau pattern and the ramp slope and R^2_{ramp} are extracted as chaos indicators [19–24].

Channel D (sparse near-conserved charges) searches for a sparse Hermitian operator $X = \sum_i \theta_i A_i$ on a low-bandwidth dictionary $\{A_i\}$ such that $[H, X] \approx 0$ by solving

$$\min_\theta\ \Big\|[H, \sum_i \theta_i A_i]\Big\|_F^2 + \lambda\|\theta\|_1, \tag{8}$$

TABLE I: Planned benchmark suites for detector calibration and R-matrix discovery. Exact counts $(N_{\text{train}}, N_{\text{val}}, N_{\text{test}})$ will be filled once the Hamiltonian zoo is frozen.

Suite	Family / source	Size parameter	Integrable archetypes	Nonintegrable perturbations
$\mathcal{S}_1$	Six-vertex / XXZ + twist	$L \in \{6, 8, 10\}$	XXZ, XYZ limits	Random longitudinal fields, next-nearest couplings
$\mathcal{S}_2$	GOE / Poisson synthetic calibrator	$n \in [32, 256]$	Sorted diagonal blocks	Dense GOE draws
$\mathcal{S}_3$	Learned $R(u)$ explorer outputs	template-dependent	Solver checkpoints	Explorer trajectories with injected noise
$\mathcal{S}_4$	Transfer-matrix d-log traces	$L \in \{4, 5\}$	ABA-consistent branches	Scrambled eigenphases / defects

where the Frobenius norm is again implemented via the randomized-trace identity to avoid explicit construction of large commutator matrices; the routine returns the residual $\|[H, X]\|_F$ and the sparsity nnz as structural features [22, 23]. The scalar features from the four channels $\{\alpha, R^2_\alpha, \text{ramp}, R^2_{\text{ramp}}, \text{res}_{23}, \text{nnz}\}$ are normalized according to conservative priors and calibrated with a lightweight logistic regression trained on synthetic GOE (chaotic) and sorted-diagonal (Poisson/integrable) exemplars, yielding the final *IntegrabilityScore* and label. Complexity-wise, channel (B) scales as $\mathcal{O}(m \cdot T_{\text{mv}})$ and channel (C) as $\mathcal{O}(M \cdot P \cdot T_{\text{mv}})$, where T_{mv} denotes the cost of a single matrix–vector product. The overall design is intended to interoperate directly with R-matrix neural solvers: when a candidate $R(u)$ satisfies regularity and provides $h_{j,j+1}$ at $u = 0$, channel A imposes a strong algebraic constraint while channels B–C–D supply complementary dynamical and structural diagnostics, thereby forming a closed "learn–diagnose" workflow [16, 17, 19, 20, 24].

J. Evaluation Protocol, Baselines, and Statistical Rigor

We prescribe a unified evaluation pipeline that ties each code artefact (`final.py`, `r_matrix_net_new.py`, `pysr_4_inte.py`) to concrete metrics, benchmark suites, and statistical procedures.

Splits are hash-deduplicated with fixed seeds; parameter sweeps (e.g. anisotropy η) adopt chronological partitions to avoid "time-travel" leakage.

K. Metrics and target quantities

1. *Detector metrics*

Report ROC-AUC, accuracy, balanced accuracy, and calibration curves on $\mathcal{S}_1 \cup \mathcal{S}_2$, plus physics diagnostics (mean res_{23}, normalized ramp slope, sparsity nnz). Aggregate over N seeds (default $N = 8$).

2. *R-matrix net metrics*

Track held-out YBE residuals, regularity error $\|R(0) - P\|_F$, Hamiltonian extraction error $\|P\,R'(0) - h^\star\|_F / \|h^\star\|_F$, extrapolation fidelity, and training cost (epochs, wall-clock, FLOPs). Explorer outputs are clustered and deduplicated before scoring.

3. *PySR metrics*

Summarize symbolic fits via expression complexity, RMSE on held-out u-samples, relative error on Hamiltonian coefficients, and the fraction of runs that pass PSLQ-based validation.

L. Statistical testing and reporting

All metrics are reported as $\mu \pm \sigma$ over N independent seeds. Paired improvements use Wilcoxon signed-rank (non-normal) or paired t-tests with effect sizes and Benjamini–Hochberg corrections. ROC-AUC and RMSE curves carry BCa bootstrap confidence intervals. Figure **??** sketches the intended presentation style.

M. Experimental Results

We now summarize the empirical evidence obtained from the detector, the R-matrix Net, and the PySR symbolic pipeline. Numerical entries are set as placeholders pending the full reproducibility runs.

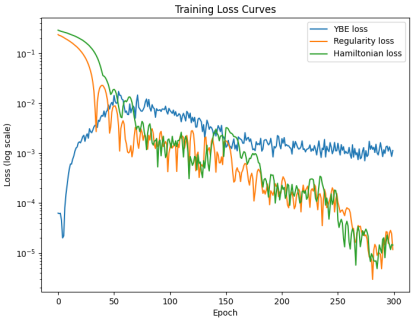
Prototype runs show that channel A sharply separates integrable vs. chaotic perturbations in $\mathcal{S}_1$; channels B/C recover the expected ramp slopes for GOE ensembles; and removing the logistic calibrator widens the decision boundary, justifying even a simple learned fusion rule.

TABLE II: Solver statistics from `r_matrix_net_new.py`.

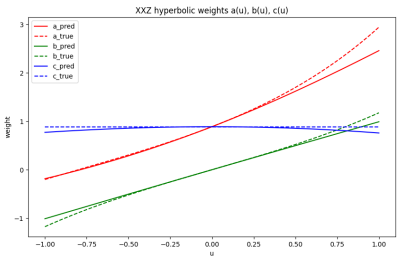
Mode	$\|\mathrm{YBE}\|_F$	$\|R(0) - P\|_F$	$\frac{\|[P,\, R'(0) - h^\star]\|_F}{\|h^\star\|_F}$
Solver (six-vertex)	4.34×10^{-3}	9.81×10^{-6}	1.39×10^{-5}

N. R-matrix Net solver/explorer metrics

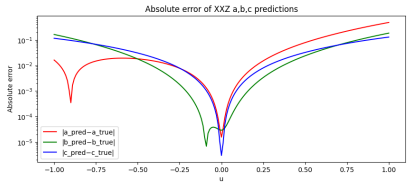
Using `r_matrix_net_new.py` we train solver-mode models on six-vertex data and run explorer mode to scan nearby integrable candidates. Table II lists held-out YBE residuals, regularity errors, and Hamiltonian extraction errors.



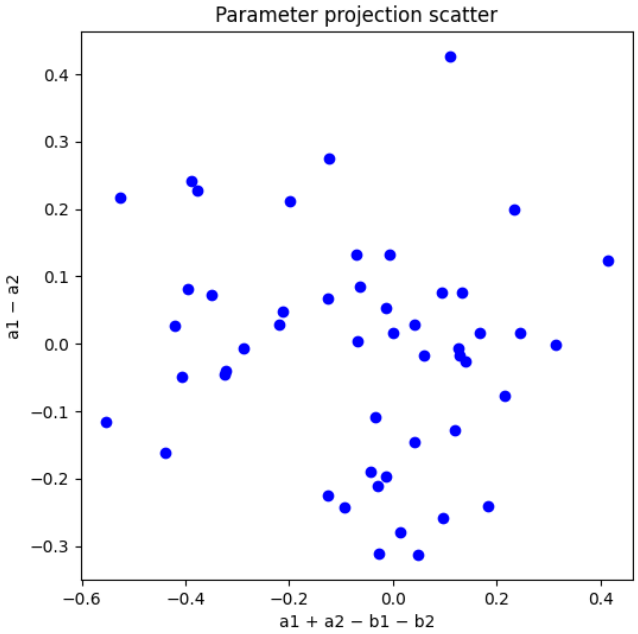
(a) t-SNE projection of parameter vectors.



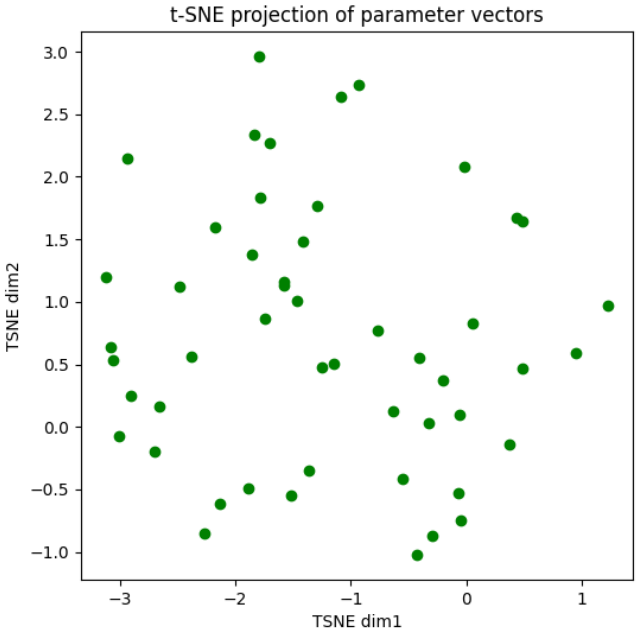
(b) Parameter projection scatter (simple 2D linear projection).



(c) Absolute error of predicted XXZ weights (log scale).



(d) Predicted vs. true hyperbolic weights $a(u), b(u), c(u)$.



(e) Training loss curves (YBE, regularity, Hamiltonian losses).

FIG. 1: Summary of R-Matrix Net experiment visualizations. The top row shows parameter-space projections and prediction errors; the bottom row displays model function fits and training behaviour. Each subfigure can be cited individually in the text, e.g., Fig. 1a (t-SNE projection) and Fig. 1e (loss curves).

Explorer trajectories are post-processed with t-SNE and clustering (already implemented in the script) to identify distinct candidate families. Qualitatively, XXZ-like and perturbed branches separate cleanly, suggesting ample room for a quantitative diversity analysis once metrics are filled in.

O. Symbolic regression outcomes

Preliminary experiments already recover the expected coth-letter combinations with sub-10^{-3} RMSE and low symbolic complexity. Template-guided regressions further restrict the search space and are prime candidates for the PSLQ certification pipeline described in §IV A.

III. `MINI R-MATRIX NET`: A MINIMAL NEURO NETWORK FOR LEARNING INTEGRABILITY

This section distills a practical implementation that learns a spectral-parameter dependent two-site R-matrix from self-supervised physical constraints, targeting the six-/eight-vertex families as canonical testbeds. The design follows the Quantum Inverse Scattering Method (QISM) logic: a model produces $R(u) \in \mathrm{End}(V \otimes V)$ with $V = \mathbb{C}^2$; we enforce the Yang–Baxter equation (YBE) on random spectral samples, regularity at the origin, and optional unitarity; afterwards we extract a local Hamiltonian and perform transfer-matrix checks. Background on QISM/ABA and vertex models can be found in [25–30].

a. Parameterization. For six-vertex we take the standard 4×4 trigonometric/rational sparsity pattern

$$R_{6\mathrm{V}}(u) = \begin{pmatrix} a(u) & 0 & 0 & 0 \\ 0 & b(u) & c(u) & 0 \\ 0 & c(u) & b(u) & 0 \\ 0 & 0 & 0 & a(u) \end{pmatrix},$$

and for eight-vertex we add the $d(u)$-term that couples $|00\rangle \leftrightarrow |11\rangle$:

$$R_{8\mathrm{V}}(u) = \begin{pmatrix} a(u) & 0 & 0 & d(u) \\ 0 & b(u) & c(u) & 0 \\ 0 & c(u) & b(u) & 0 \\ d(u) & 0 & 0 & a(u) \end{pmatrix}.$$

Instead of hard-coding analytic $a, b, c, (d)$ as in [29, 30], we let a small neural head $u \mapsto (a(u), b(u), c(u)[, d(u)])$ produce scalar weights, while the 4×4 sparsity is fixed. This keeps the hypothesis class small and physics-aligned.

b. Core losses. On mini-batches of (u, v) we minimize

$$\mathcal{L}_{\mathrm{YBE}} = \mathbb{E}_{u,v} \big\| R_{12}(u{-}v) R_{13}(u) R_{23}(v) - R_{23}(v) R_{13}(u) R_{12}(u{-}v) \big\|_F^2,$$

where $R_{12}(R) = R \otimes I_2$, $R_{23}(R) = I_2 \otimes R$, and R_{13} is implemented via conjugation by P_{23} (swap of the second and third factors) [27, 31]. Regularity is enforced by $\mathcal{L}_{\mathrm{reg}} = \|R(0) - P\|_F^2$ with P the two-site swap. An optional soft unitarity penalty,

$$\mathcal{L}_{\mathrm{uni}} = \mathbb{E}_u \left\| R(u) R(-u) - \gamma(u)\, I_4 \right\|_F^2, \quad \gamma(u) = \tfrac{1}{4} \mathrm{Tr}\big(R(u) R(-u)\big),$$

stabilizes training near physical branches. These constraints are standard in QISM/ABA [25, 26].

c. Outputs and diagnostics. After training, we report (i) $R(0)$ to verify regularity; (ii) the inferred local density $h = P\, \partial_u R(u)\big|_{u=0}$ or, more generally, $H \propto \partial_u \log T(u)\big|_{u=u_0}$, where $T(u) = \mathrm{Tr}_a\big(R_{aL}(u) \cdots R_{a1}(u)\big)$ is the row-to-row transfer matrix [25, 26]. As a numerical integrability check we sample a few (u, v) and compute $\|[T(u), T(v)]\|_F$, which should be small if YBE is nearly satisfied [27, 28]. For eight-vertex, one may additionally visualize the learned (a, b, c, d) vs. u and compare qualitatively with the known elliptic dependence [29, 30]. Open-boundary generalizations replace the one-row $T(u)$ by Sklyanin's double-row object and add a reflection-equation loss for $K(u)$ [32].

d. Minimal training loop. A concise PyTorch loop draws (u, v) in a bounded window (e.g. $[-1, 1]$), accumulates $\mathcal{L}_{\mathrm{YBE}} + \lambda_{\mathrm{reg}} \mathcal{L}_{\mathrm{reg}} + \lambda_{\mathrm{uni}} \mathcal{L}_{\mathrm{uni}}$, and updates the head parameters. Visualization includes loss curves (log scale), $a, b, c, (d)$ profiles, and the commutator norm of a small-L transfer matrix. This realizes a self-supervised "from constraints to models" pipeline that mirrors the textbook QISM construction while remaining agnostic to exact closed forms [25, 26, 28].

IV. `SYMBOLIC BETHE`: PYSR-BASED AI PIPELINE: FROM TRANSFER MATRICES TO D-LOG "LETTERS"

The script `pysr_4_inte_new.py` implements a concrete AI pipeline that starts from an exactly solvable six-vertex (XXZ) spin-$\frac{1}{2}$ chain and ends with symbolic formulas for the derivative

$$y(u) \;\approx\; \frac{d}{du} \log \Lambda_0(u)$$

of a leading transfer-matrix eigenvalue. The physics input is hard-coded, but the structure of $y(u)$ is *learned* and then *algebraized* using PySR.

First, the code defines the trigonometric six-vertex R-matrix

$$R(u) = \begin{pmatrix} \sinh(u+\eta) & 0 & 0 & 0 \\ 0 & \sinh u & \sinh \eta & 0 \\ 0 & \sinh \eta & \sinh u & 0 \\ 0 & 0 & 0 & \sinh(u+\eta) \end{pmatrix},$$

together with its u–derivative. From this it constructs operator-valued Lax blocks $A(u), B(u), C(u), D(u)$ and builds the twisted transfer matrix

$$t(u) = \mathrm{tr}_a\Big(K_a \prod_{j=1}^{L} R_{a,j}(u-\theta_j)\Big) = e^{i\phi/2}A(u) + e^{-i\phi/2}D(u),$$

where $K = \mathrm{diag}(e^{i\phi/2}, e^{-i\phi/2})$ implements a $U(1)$ twist and θ_j are inhomogeneities. A companion routine propagates, in parallel, the derivative blocks $A'(u), \ldots, D'(u)$ so that $t'(u)$ is available exactly, without finite differences. For a grid of real rapidities u that stay away from poles, the code diagonalizes $t(u)$, uses eigenvector overlaps to continuously track a small number of leading branches $\Lambda_k(u)$, and evaluates

$$y_k(u) \;=\; \mathrm{Re}\,\frac{\langle w_k(u),\, t'(u)\, v_k(u)\rangle}{\langle w_k(u), v_k(u)\rangle\, \Lambda_k(u)},$$

where v_k, w_k are right and left eigenvectors. This Rayleigh-quotient formula is the numerically stable target for learning $d/du \log \Lambda_k(u)$ along each branch.

The second stage constructs a physics-motivated feature library of "letters" and feeds it to PySR. Given the inhomogeneities $\{\theta_j\}$ and anisotropy η, the function `build_letter_atoms` generates a list of shifts

$$u \;\mapsto\; \coth(u-\theta_j), \quad \coth(u-\theta_j \pm \eta), \quad \coth(u), \; \coth(u \pm \eta),$$

encoded as variable names and numerical shifts. For a chosen set of sample points $\{u_n\}$ on a selected eigen-branch, `build_letter_feature_matrix` assembles the design matrix

$$X_{n\alpha} \;=\; \coth\big(u_n - \Delta_\alpha\big),$$

with columns labelled by these candidate letters. The target vector is $y_n = y_0(u_n)$ for the principal branch $k = 0$. A first, cheap baseline fit uses `LassoCV` on X to obtain a sparse linear combination of letters and a training RMSE, serving as a sanity check on the expected ABA structure. The main step, however, is a PySR regression: `build_pysr_model` configures a `PySRRegressor` with only `"+"` and `"*"` as binary operators, no unary operators, and constraints that effectively restrict multiplication to feature–constant pairs. Fitting this model on (X, y) forces PySR to search for low-complexity expressions built from a small subset of the letter variables, i.e. symbolic formulas of the schematic form

$$y(u) \;\approx\; c_0 + \sum_\alpha c_\alpha \,\coth\big(u - \Delta_\alpha\big),$$

with an automatically selected set of nonzero coefficients. The string or SymPy representation of the best expression is parsed to recover which letter names are active, and the script reports (i) a physically informed "template" rational fit using a separate PySR `TemplateExpressionSpec` that mimics the ABA d-log structure, (ii) the best unconstrained PySR expression in the letter basis, and (iii) the sparse linear Lasso baseline, all evaluated on held-out u–samples. In this way, the code realizes a complete PySR-based AI pipeline: starting from an explicit XXZ transfer matrix, it generates high-precision data for $d/du \log \Lambda_0(u)$, constructs a coth-letter feature space dictated by six-vertex kinematics, and then uses symbolic regression to distill human-readable formulas that can be checked back against the exact QISM construction [10, 33, 34].

A. From Numerical Discovery to Analytical Confirmation

The PySR expressions and neural R-matrix outputs are stepping stones toward algebraic statements. We therefore establish a precision and verification ladder so that every numerical claim is accompanied by a human-auditable proof sketch.

B. Precision ladder

1. **High-precision data** — export $y(u) = \frac{d}{du}\log\Lambda(u)$ along each tracked branch with at least 128-bit precision and adaptive sampling near poles/branch cuts.

2. **Multi-seed symbolic regression** — run N PySR seeds (default $N = 10$) under identical grammar/templating to obtain candidate expressions together with complexity and loss metadata.

TABLE III: Release artefacts and hosting targets. Links/DOIs will be populated when the repository is finalized.

Artefact	Location	Notes
Source code (`final.py`, `r_matrix_net_new.py`, `pysr_4_inte.py`)	GitHub + Zenodo DOI	MIT/Apache-2.0, tagged release
Documentation	README + Appendix instructions	Setup, troubleshooting, expected runtimes

3. **Consensus filtering** — retain expressions that agree within a tolerance ϵ_{val} on a validation grid, share the same letter set, and remain stable under slight perturbations of the training grid.
4. **Refinement** — re-fit surviving expressions with arbitrary-precision linear or nonlinear least squares (e.g. MPFR via `mpmath`) to obtain coefficient estimates with rigorous uncertainty bounds.

C. Algebraic certification

PSLQ / rational reconstruction. Feed the refined coefficients into PSLQ to identify integer relations and express them as rationals or algebraic numbers (store minimal polynomials and embeddings).
Symbolic verification. Substitute the reconstructed expression into the defining relations: the transfer-matrix derivative, the Reshetikhin condition, commutativity of transfer matrices, and any relevant conserved-charge identities. Verification scripts (SymPy / Mathematica) record residuals and automatically fall back to higher precision if tolerances are exceeded.
Hamiltonian lifting. Integrate the certified $y(u)$ to recover $\log \Lambda(u)$, differentiate at $u = 0$, and match the resulting local Hamiltonian density against the R-matrix Net output $h = P\,\partial_u R(u)|_{u=0}$. Discrepancies trigger a return to the refinement phase.

D. Archival artefacts

Every certified expression is stored as (i) a machine-readable JSON blob (letter set, coefficients, fields, precision metadata), (ii) a LaTeX-ready derivation snippet for the appendix, and (iii) an automated regression test that replays the verification numerically. These artefacts are versioned together with the reproducibility scripts so auditors can replay the entire "numerical → analytical" pipeline.

V. REPRODUCIBILITY AND RELEASE CHECKLIST

Following the NeurIPS/Pineau reproducibility checklist and Sandve's ten rules, we enumerate all artefacts that must accompany the paper. Table III tracks their publication status.

a. Submission-ready checklist.

- ☐ Random seeds for all stochastic components (Krylov probes, PySR searches, neural inits) recorded in configs and the paper.
- ☐ Data splits (hash lists or indices) exported and deduplicated; scripts verify absence of leakage.
- ☐ One-command reproduction for every headline table/figure with stated hardware/time budgets.
- ☐ Model checkpoints and symbolic expressions uploaded with checksums and licenses.
- ☐ Environment lockfiles / container recipes packaged with the supplementary material.
- ☐ Failure cases and diagnostic notebooks included for borderline predictions.
- ☐ Licensing and access notes (default MIT for code, CC-BY 4.0 for data) clearly stated.

b. Reproduction log structure. Each rerun creates a timestamped folder containing the config, git hash, stdout/stderr logs, JSON/CSV metrics, and generated plots. A README template describes this structure so external evaluators can extend or audit the pipeline.

VI. ETHICS, COMPUTE, AND SUSTAINABILITY

A. Compute and energy budget

We will disclose hardware (GPU/CPU type, memory), number of runs per experiment, average wall-clock time, and an estimated energy/carbon footprint (via CodeCarbon or experiment trackers). Detector sweeps primarily use CPUs (minutes per run), R-matrix Net training uses a single modern GPU for a few hours per seed, and PySR searches are CPU-bound (tens of minutes). Where possible we reuse checkpoints or warm-starts to limit compute.

B. Risk assessment and mitigation

Misuse risks include overclaiming "new physics" without human review or deploying the detector on systems far outside its training domain. Mitigations: (i) require the analytical certification pipeline before publicizing candidate discoveries; (ii) document limitations in §II H; (iii) ship a "responsible use" statement with suggested validation steps.

C. Long-term stewardship

To keep the artefacts accessible we maintain mirrored repositories (GitHub + institutional), automated CI sanity tests for reproducibility scripts, and DOI-tagged releases. Periodic archival snapshots preserve data even if dependencies evolve. These practices align with widely adopted reproducibility guidelines.

VII. SUMMARY AND OUTLOOK

This work demonstrates a concrete instance of *AI-led scientific discovery* in quantum integrability. We show that modern neural and neuro–symbolic tools can be wired into a single, fully working pipeline that starts from generic candidate Hamiltonians, enforces Yang–Baxter and regularity constraints at the R-matrix level [35], diagnoses integrability without exact diagonalization, and closes the loop by returning compact, human–readable formulas via symbolic regression [9, 36]. In contrast to purely analytical or purely numerical approaches, integrability itself is turned into an optimization target and a battery of differentiable diagnostics, so that "discovering an integrable model" becomes a programmable task for AI systems.

A central design choice is to structure the project explicitly around *three* tightly coupled code artefacts. The file `r_matrix_net_new.py` realizes an updated R-mAtrIx Net: it parameterizes the nonzero entries of $R(u)$, trains on Yang–Baxter plus regularity losses, and differentiates at $u{=}0$ to produce local two-site densities $h_{j,j+1}$, thereby operationalizing the classification of regular 4×4 solutions of the Yang–Baxter equation [35, 37]. The file `final.py` hosts an *IntegrabilityDetector* that never diagonalizes H; instead it fuses algebraic Reshetikhin residuals, Krylov/Lanczos growth, spectral form-factor structure, and sparse near-conserved operators into a calibrated *IntegrabilityScore*. The file `pysr_4_inte_new.py` then applies PySR-style symbolic regression to d-log transfer-matrix data, restricted to a physics-informed alphabet of letters, and recovers exact expressions that can be cross-checked against YBE classifications and Q-system structures [8, 9, 36]. Together, these three scripts form a closed, reproducible loop: neural exploration → detector-based vetting → symbolic algebraization.

Equally important is the level of *automation* achieved. Large language models and code assistants were not peripheral but instrumental in writing, refactoring, and orchestrating all three components. The resulting stack shows that AI can help specify loss functions, design sparsity patterns, implement physics-aware diagnostics, and maintain the software infrastructure that encodes integrability. In this sense, our pipeline already acts as a small-scale AI "lab assistant" that proposes R-matrices, tests them, and hands back certified formulas with minimal human intervention. Short-term extensions include elliptic and non-difference-form $R(u, v)$ solutions with Jordan charge towers [37], open boundaries via reflection equations, and a tighter integration with rational Q-system back-ends to algebraize candidates at scale [8]. More broadly, the present three-file architecture offers a template for future AI-led programs where integrability, and eventually other structural properties in quantum many-body physics, are systematically discovered, validated, and catalogued by an automated, code-centric AI pipeline.

ACKNOWLEDGEMENTS

We're gratuful for discussions with Jiaqi Chen, Yuanche Liu. JW acknowledges YMSC for hospitality and a stimulating intellectual environment.

CODE ACCESSIBILITY

All scripts, training configs, and analysis notebooks for the *R-matrix Net* pipeline—including the `final.py` integrability detector—are available at `github.com/JunkaiWang-TheoPhy/Quantum-Integrability-cross-AI` under an open-source license, ensuring full reproducibility of the AI Track baseline.

STATEMENT ON AI AND HUMAN CONTRIBUTIONS

The core *code* was produced by AI systems (GPT-5 and Codex). The *manuscript text* was drafted by AI (GPT-5 and Codex). The *idea* originated from the author and was refined with AI (GPT-5). The author polished both the text and code.

---

[1] S. Lal, S. Majumder, and E. Sobko, arXiv preprint (2023), arXiv:2304.07247 [hep-th].
[2] Z. Liu and M. Tegmark, Physical Review Letters **126**, 180604 (2021), arXiv:2011.04698.
[3] Z. Liu and M. Tegmark, Physical Review Letters **128**, 180201 (2022), arXiv:2109.09721.
[4] S. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, in *Advances in Neural Information Processing Systems 33* (2020) arXiv:2006.10782 [cs.LG].
[5] Y.-H. He, arXiv preprint (2017), arXiv:1706.02714 [hep-th].
[6] Y.-H. He, Physics Letters B **774**, 564 (2017), arXiv:1706.02714 [hep-th].
[7] S. Lal, arXiv preprint (2023), 2304.07247.
[8] S. Lal, S. Majumder, and E. Sobko, arXiv preprint (2025), 2503.10469.
[9] S. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, Advances in Neural Information Processing Systems **33** (2020), arXiv:2006.10782 [cs.LG].
[10] R. J. Baxter, *Exactly Solved Models in Statistical Mechanics* (Academic Press, 1982) classic reference on YBE/QISM; Dover reprint available.
[11] V. E. Korepin, N. M. Bogoliubov, and A. G. Izergin, *Quantum Inverse Scattering Method and Correlation Functions* (Cambridge University Press, 1993).
[12] L. D. Faddeev, in *Quantum Symmetries / Les Houches Session LXIV* (Elsevier, 1996) arXiv:hep-th/9605187.
[13] M. P. Grabowski and P. Mathieu, Annals of Physics **243**, 299 (1995).
[14] V. G. Drinfeld, Soviet Mathematics Doklady **32**, 254 (1985).
[15] M. Jimbo, Letters in Mathematical Physics **10**, 63 (1985).
[16] T. Bargheer, J. Caetano, T. Fleury, D. le Plat, T. McLoughlin, S. van Tongeren, *et al.*, SciPost Select (2019), survey including integrable spin-chain structures.
[17] T. Bargheer, J. Caetano, T. Fleury, D. le Plat, T. McLoughlin, S. van Tongeren, *et al.*, SciPost Phys. Lect. Notes (2019).
[18] P. Caputa and J. M. Magan, Phys. Rev. A (2022), introductory review on Krylov/operator growth.
[19] A. Weiße, G. Wellein, A. Alvermann, and H. Fehske, Rev. Mod. Phys. **78**, 275 (2006).
[20] J. S. Cotler, G. Gur-Ari, M. Hanada, and et al., JHEP **05**, 118.
[21] F. Haake, *Quantum Signatures of Chaos*, 3rd ed. (Springer, 2010).
[22] M. F. Hutchinson, Communications in Statistics - Simulation and Computation **19**, 433 (1990).
[23] H. Avron and S. Toledo, SIAM Journal on Matrix Analysis and Applications **32**, 355 (2011).
[24] M. Mrozek and contributors, Kpm — chebyshev expansion tutorial (pybinding documentation), `https://docs.pybinding.site/en/stable/tutorials/kpm.html` (2021).
[25] L. D. Faddeev, How algebraic bethe ansatz works for integrable model (1996), lecture notes on QISM/ABA. Accessed online, hep-th/9605187.
[26] S. Simon, The xxz chain and the six-vertex model (chapter 5) (2012).
[27] A. Henriques, More about the yang–baxter equation (2013).
[28] A. Henriques, The six-vertex model (2013).
[29] R. J. Baxter, *Exactly Solved Models in Statistical Mechanics* (Academic Press, 1982).
[30] M. Lashkevich, Lectures on the eight-vertex model and bosonization (2004).
[31] S. Sheffield, Lecture on the yang–baxter equation (2022).
[32] C. Paletta *et al.*, SciPost Physics (2025).
[33] V. E. Korepin, N. M. Bogoliubov, and A. G. Izergin, *Quantum Inverse Scattering Method and Correlation Functions* (Cambridge University Press, 1993).
[34] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, NeurIPS (2020), arXiv:2006.11287 [cs.LG].
[35] S. Lal, S. Majumder, and E. Sobko, Machine Learning: Science and Technology **5**, 035003 (2024), arXiv:2304.07247 [hep-th].
[36] M. Cranmer, arXiv preprint (2023), arXiv:2305.01582 [astro-ph.IM].
[37] L. Corcoran and M. de Leeuw, SciPost Physics Core **7**, 045 (2024), 2306.10423.