Aalto University Department of Applied Physics

PHYS-EV0007 — Machine Learning from and for Quantum Science

Topic: Hamiltonian learning of a fermionic chain
via nonlocal impurity tomography

Based on G. Lupi and J. L. Lado, Phys. Rev. Applied **23**, 054077 (2025)

Anouar Moustaj
anouar.moustaj@aalto.fi

Topic nr 3
Project nr 1

Project description

Understand and apply the Hamiltonian-learning methodology of to a *minimal* quantum-magnet problem. Concretely: simulate a 1D spin-orbit-coupled tight-binding chain with a local magnetic impurity, compute the spectral function, generate a training dataset by sweeping the microscopic parameters, and train a neural network to *learn back* those parameters from the impurity spectral fingerprints.

Learning outcomes

- Gain fluency with the Hamiltonian-learning inverse problem on a single-particle model: dataset construction, NN architecture choice, two-step noise retraining, fidelity evaluation.
- Understand how a local magnetic impurity scatters Bloch states and imprints $2k_F$-type oscillations on the local DOS (a single-particle analogue of *quasiparticle interference*).
- Understand how the spatial-and-frequency-resolved spectral fingerprint of the impurity can be inverted to recover the parent Hamiltonian.
- Learn how to model a single-particle tight-binding model

Presentation

The presentation should cover the *full* scope of the paper (both the fermionic single-particle case and the many-body spin-chain case), independently of the project simulations, which will be restricted to the fermionic problem.

- Explain the formalism used in the paper: tight-binding / Green's-function methods for the fermionic case, and DMRG / MPS methods for the many-body spin-chain case.
- Understand the role of impurities in both cases, as local perturbations whose imprint on the ground state provides access to the nonlocal response of the parent Hamiltonian.
- Explain the implementation of the Hamiltonian-learning pipeline.
- Explain the advantages and limitations of the method.

Simulations

1. **Building the fermionic chain and simulating $A(n,\omega)$.** Using pyqula, construct the Rashba tight-binding chain of length N (e.g. $N = 150$ as in the paper) with first-neighbour hopping t_1, second-neighbour hopping t_2, chemical potential μ, and Rashba SOC λ_R,

$$H_{\text{chain}} \;=\; t_1 \sum_{\langle i,j\rangle,s} c^\dagger_{i,s} c_{j,s} + t_2 \sum_{\langle\langle i,j\rangle\rangle,s} c^\dagger_{i,s} c_{j,s} + \mu \sum_{i,s} c^\dagger_{i,s} c_{i,s} + i\lambda_R \sum_{\langle i,j\rangle,s,s'} [\hat{z}\times(\vec{r}_i - \vec{r}_j)]\vec{\sigma}_{ss'}\, c^\dagger_{i,s} c_{j,s'}. \tag{1}$$

   Fix $t_1 = 1$ as the energy scale. Now add a local magnetic impurity at the chain centre with strength δ (take $\delta = 0.4$ as in the paper). In the tight-binding matrix this is just a 2×2 Zeeman-like block on the centre-site diagonal,

$$H \;=\; H_{\text{chain}} \;+\; H_{\text{imp}}, \qquad H_{\text{imp}} \;=\; \delta \sum_{s,s'} \sigma^z_{ss'}\, c^\dagger_{i_0,s} c_{i_0,s'} \;=\; \delta\big(n_{i_0,\uparrow} - n_{i_0,\downarrow}\big), \tag{2}$$

   where i_0 labels the central site. Solve the resulting quadratic problem with a standard Green's-function embedding algorithm in pyqula and extract the local spectral function at site n,

$$A(n,\omega) \;=\; \sum_s \langle\Omega|\, c_{n,s}\, \delta(\omega - (H - E_0))\, c^\dagger_{n,s}\, |\Omega\rangle \;=\; -\frac{1}{\pi}\, \mathrm{Im}\big[G^{(R)}_{nn}(\omega + i\eta)\big], \tag{3}$$

   with a small broadening $\eta \sim 10^{-2}\, t_1$.

   a. For a clean, uniform chain (no impurity) plot $A(n,\omega)$ on the (n,ω) plane and verify that it is site-independent in the bulk, with a DOS that reflects the 1D band structure of the t_1, t_2, μ, λ_R Hamiltonian.

   b. Turn the impurity on and plot $A(n,\omega)$ on the (n,ω) plane; identify the impurity-induced Friedel-like oscillations and their characteristic wavevector.

   c. Compute the momentum-resolved spectral function $\tilde{A}(k,\omega)$, and plot it on the (k,ω) plane. Verify that the bright features track the quasiparticle-interference pattern expected from the unperturbed dispersion.

   d. Benchmark: for $\lambda_R = 0$, $t_2 = 0$ the dispersion is the simple cosine band $\epsilon(k) = 2t_1 \cos k + \mu$; check your numerics against this analytic limit.

2. **Training dataset for Hamiltonian learning.** Generate a dataset of

$$\{(\{A(n,\omega_q)\}_{n,q},\, \{\tilde{A}(k_p,\omega_q)\}_{p,q},\, (\mu, t_2, \lambda_R))\}$$

   pairs by sampling the microscopic parameters on a fixed range.

   a. Evaluate the spectral function at three representative frequencies ω_r (in units of $t_1/\hbar$) and stack the real-space slices $\{A(n,\omega_r)\}_{r=1,2,3}$ together with their Fourier transforms $\{\tilde{A}(k,\omega_r)\}_{r=1,2,3}$ into a single feature vector.

   b. Sample N_{train} training and $N_{\text{test}} < N_{\text{train}}$ test parameter triplets uniformly in the above ranges.

   c. Save the dataset $\{(\text{features}, (\mu, t_2, \lambda_R))\}$ to disk as `.npz`.

   d. Visualise a handful of $(A(n,\omega), \tilde{A}(k,\omega))$ pairs across the grid and confirm that the features evolve smoothly with (μ, t_2, λ_R) and in particular that the subtle Rashba SOC is more visible in $\tilde{A}(k,\omega)$ than in $A(n,\omega)$.

e. Estimate the wall-clock time per Green's-function solve and plan your grid resolution accordingly.

3. **Training a neural-network regressor for (μ, t_2, λ_R).** Train a fully-connected feed-forward neural network that inverts the spectral fingerprint,

$$f_\theta : \ (\{A(n,\omega_r)\}_{n,r},\, \{\tilde{A}(k,\omega_r)\}_{k,r}) \ \longmapsto \ (\mu,\, t_2,\, \lambda_R), \tag{4}$$

with mean-squared-error loss and the Adam optimizer (choose appropriate learning rates and batch sizes). You can get inspiration from the paper's architecture. Start from a lighter version (a handful of `Dense(256)` layers) and only scale up if needed.

   a. Split into train / validation / test; standardise both inputs and outputs.

   b. Train on the *clean* data and report learning curves.

   c. Evaluate on the test set: plot predicted vs. true μ, t_2, λ_R and the Pearson-correlation fidelity

$$\mathcal{F}_\Lambda \ = \ \frac{\langle \Lambda^{\mathrm{pred}} \Lambda^{\mathrm{true}} \rangle - \langle \Lambda^{\mathrm{pred}} \rangle \langle \Lambda^{\mathrm{true}} \rangle}{\sqrt{\mathrm{var}(\Lambda^{\mathrm{pred}})\, \mathrm{var}(\Lambda^{\mathrm{true}})}}, \tag{5}$$

   with $\Lambda \in \{\mu, t_2, \lambda_R\}$.

4. **Two-stage noise retraining and diagnostics.**

   a. **Electronic / background noise.** Re-use the clean-trained model and resume training for a fixed number of epochs on data augmented with additive Gaussian noise on the spectral function,

$$A(n,\omega)_{\mathrm{ElecNoise}} \ = \ A(n,\omega) + \mathcal{N}(0, \Delta_E^2), \tag{6}$$

   for two levels $\Delta_E^2 \in \{0.05, 0.10\}$. Evaluate fidelity $\mathcal{F}_\Lambda$ for each parameter and each noise level; plot $\mathcal{F}_\Lambda$ vs. noise level and comment on which parameter is most fragile.

   b. **Scaling / setpoint-current noise.** Apply a multiplicative scaling noise on the test spectra,

$$A(n,\omega)_{\mathrm{ScaNoise}} \ = \ A(n,\omega)(1 + W(n_s)), \tag{7}$$

   with W drawn uniformly in $[-0.5, 0.5]$, and sweep the overall noise amplitude $W \in [0, 3]$. Report $\mathcal{F}_\Lambda(W)$ for the three models {clean / retrained at $\Delta_E^2 = 0.05$ / retrained at $\Delta_E^2 = 0.10$} and reproduce qualitatively the trends in Fig. 4(b) of the paper.

   c. **Out-of-distribution testing.** Evaluate the model on parameter triplets just outside the training ranges and on spectra generated with small broadenings η different from the training value, and comment on the extrapolation behaviour.

5. **Performance benchmarking.**

   a. Measure the wall-clock time of the Green's-function forward solve as a function of chain length N and of the NN training as a function of dataset size.

   b. Report the final fidelities $\mathcal{F}_\mu$, $\mathcal{F}_{t_2}$, $\mathcal{F}_{\lambda_R}$ as a function of training-set size; identify the bottleneck (forward solver vs. training).

   c. Comment on what would change if you wanted to learn a fourth parameter jointly (e.g. a third-neighbour hopping t_3 or a Zeeman field B), and how the diagnostics of step 4 would need to be extended.

Deliverables

- A concise report in the form of a presentation.
- A repo with organized code, well-documented, and a notebook containing a reproducible end-to-end run (fermionic-chain spectral-function dataset generation → NN training → two-stage noise retraining → diagnostics) for a modest chain length.

Working practices and tools

You are strongly encouraged to recycle existing material. You are expected to **read, reuse, and adapt** existing reference implementations pipelines rather than re-derive or re-implement every algorithmic detail from scratch; the pedagogical goal of this project is to *understand and apply* the method, not to reproduce boilerplate. Any code you borrow must be clearly attributed in your repository (e.g. in a `README.md` or as in-source comments) and integrated cleanly with your own contributions. *Use of LLMs (ChatGPT, Claude, Gemini, . . .) is permitted and encouraged for onboarding.* In particular, LLMs are very effective at:

- explaining unfamiliar programming syntax and idioms;
- summarising sections of the paper or cross-referencing related literature;
- debugging installation issues, bookkeeping, and programming recommendations;
- generating boilerplate (plotting scripts, parameter sweeps, unit tests, diagnostic helpers).

You remain, however, fully responsible for the correctness, clarity, and scientific content of what you submit: LLM output must be checked, understood, and, where appropriate, cited. Treat an LLM as a capable but occasionally wrong collaborator, not as an oracle.

Programming language: Python.

Packages required: `numpy`, `scipy`, `matplotlib`, PyTorch (or TensorFlow / scikit-learn)

References

- G. Lupi and J. L. Lado, Phys. Rev. Applied **23**, 054077 (2025); preprint arXiv:2412.07666;
- G. Lupi, companion GitHub repository.
- J. L. Lado, pyqula — Python library for tight-binding simulations of single-particle problems.