Aalto University Department of Applied Physics

PHYS-EV0007 — Machine Learning from and for Quantum Science

Project: Extracting the spin–orbit coupling of a molecular quantum magnet from setpoint-dependent STM-IETS via machine learning

Based on G. Lupi et al., arXiv:2601.19371 (2026)

Anouar Moustaj
anouar.moustaj@aalto.fi

Topic nr 3
Project nr 3

Project description

Understand and apply the molecular Hamiltonian-learning methodology of arXiv:2601.19371 to a *minimal* problem. Concretely: build a small multi-orbital molecular Hamiltonian, simulate its STM-IETS spectra as a function of the tip setpoint, generate a training dataset by sweeping the spin–orbit coupling λ, and train a neural network to *learn back* λ from the setpoint-dependent spectra.

Learning outcomes

- Gain fluency with the Hamiltonian-learning inverse problem on a multi-orbital molecular system: dataset construction, NN architecture choice, training / validation, fidelity evaluation.
- Understand minimal multi-orbital models of molecular quantum magnets (crystal-field splitting, local Coulomb interaction, spin–orbit coupling) and how to simulate their STM-IETS spectra.
- Understand the experimental notion of a "setpoint" and how the tip-induced electrostatic environment enters the Hamiltonian, and why the setpoint dependence is an informative probe of λ.
- Learn how to model a small fermionic multi-orbital Hamiltonian by exact diagonalisation in Fock space with DMRGPy, and how to compute its dynamical correlators.

Presentation

- Explain the formalism used in the paper: multi-orbital molecular Hamiltonians (crystal-field splitting, on-site Coulomb interactions, spin–orbit coupling) and STM-IETS observables.
- Understand the role of the setpoint as a local perturbation whose imprint on the spectrum provides access to the parameters of the parent Hamiltonian.
- Explain the implementation of the Hamiltonian-learning pipeline.
- Explain the advantages and limitations of the method.

Simulations

The simulation exercise is a *single-target* reduction of the four-parameter problem of the paper: all nuisance parameters are held fixed, and the network is trained to recover only the spin–orbit coupling λ_{SOC} from setpoint-dependent spectra. The forward solver is the multi-orbital many-body model of the paper, and you are strongly encouraged to reuse the reference implementation `src/helper.py` of the companion GitHub repository (`hp.calculate_hamiltonian`, `hp.get_orbital_cotunneling`, `hp.get_orbital_spinflip`, `hp.get_spinflip`).

1. **Building the molecular Hamiltonian and simulating the dynamical correlator.** Construct the multi-orbital many-body model for a single Fe-phthalocyanine molecule on a ferroelectric SnTe substrate: the full Fe $3d$ shell (five orbitals $\{d_{z^2},\, d_{xz},\, d_{yz},\, d_{x^2-y^2},\, d_{xy}\}$) at fixed filling $N_e = 6$ (Fe^{2+}),

$$H \;=\; H^{0}_{\mathrm{CF}} \;+\; H^{\mathrm{tip}}_{\mathrm{CF}}(\varepsilon) \;+\; H_{\mathrm{int}} \;+\; H_{\mathrm{SOC}}(\lambda_{\mathrm{SOC}}), \tag{1}$$

with

$$H^{0}_{\mathrm{CF}} = \sum_{i,j,s} t_{ij}\, c^{\dagger}_{i,s} c_{j,s}, \qquad H^{\mathrm{tip}}_{\mathrm{CF}}(\varepsilon) = \varepsilon \sum_{s} c^{\dagger}_{d_{z^2},s}\, c_{d_{z^2},s}, \tag{2}$$

$$H_{\mathrm{int}} = -U\,\vec{S}^{2} \;-\; J\,\vec{L}^{2}, \qquad H_{\mathrm{SOC}} = \lambda_{\mathrm{SOC}} \sum_{i,j,\alpha,s,s'} \ell^{\alpha}_{ij}\, \sigma^{\alpha}_{ss'}\, c^{\dagger}_{i,s} c_{j,s'}. \tag{3}$$

The single-particle matrix t_{ij} encodes a fixed crystal-field splitting $\Delta_0 = 1.5\,\mathrm{eV}$ together with a ferroelectric distortion τ that lifts the (d_{xz}, d_{yz}) degeneracy; fix τ to a single representative value (e.g. $\tau = 0.1\,\mathrm{eV}$, $\theta = 0$). The tip setpoint enters as a Stark shift ε *on the* d_{z^2} *orbital only*, reflecting the fact that the STM tip couples predominantly to d_{z^2}. The interactions are taken in the simplified form $-U\,\vec{S}^2 - J\,\vec{L}^2$ ($U = 4.0\,\mathrm{eV}$, $J = 0.05\,\mathrm{eV}$), which enforces the high-spin $S = 2$ ground state at a modest cost in the many-body solver. Build H with `hp.calculate_hamiltonian` and diagonalise in Fock space (exact diagonalisation).

   a. Scan a fixed setpoint window $[\varepsilon_{\min}, \varepsilon_{\max}]$ (e.g. $[-0.4, -0.3]\,\mathrm{eV}$) at N_ε values (e.g. $N_\varepsilon = 80$). For each ε, compute the tip-projected dynamical correlator $S(\omega;\varepsilon) \propto \mathrm{d}^2 I/\mathrm{d}V^2$ on a frequency grid $\omega \in [0, 0.2]\,\mathrm{eV}$ (e.g. $N_\omega = 251$), as the sum of three channels: orbital cotunneling through d_{z^2}, orbital spin-flip through d_{z^2}, and local spin-susceptibility $\langle S^{\alpha}_{d_{z^2}}; S^{\alpha}_{d_{z^2}} \rangle$.

   b. Cumulatively integrate $S(\omega;\varepsilon)$ along ω to obtain the experimentally-relevant $\mathrm{d}I/\mathrm{d}V(\omega;\varepsilon)$, which will be the input to the NN.

   c. Plot $\mathrm{d}I/\mathrm{d}V$ as a 2D map on the (ω, ε) plane for a few representative λ_{SOC} values; identify the inelastic steps whose positions track λ_{SOC}-induced magnetic anisotropy.

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

$$\{(\mathrm{d}I/\mathrm{d}V(\omega_q, \varepsilon_p),\ \lambda_{\mathrm{SOC}})\}$$

pairs by sampling λ_{SOC} uniformly in $[0,\, 0.15]\,\mathrm{eV}$, at fixed $(\tau, \varepsilon_{\min}, \varepsilon_{\max}, \Delta_0, U, J)$.

   a. Preprocess each sample following the paper's protocol: excise the central bias region $|\omega| < 30\,\mathrm{meV}$ (Kondo-dominated), subtract the residual DC offset, max-normalise per sample.

   b. (Optional, recommended.) Stack the (ε, ω) map into a single feature vector and reduce its dimension with PCA (e.g. 200 components) fitted on the training set, following `data_loader.py`.

c. Sample N_{train} training and $N_{\text{test}} < N_{\text{train}}$ test values of λ_{SOC} uniformly in the above range; save the dataset $\{(\text{features}, \lambda_{\text{SOC}})\}$ to disk as `.npz`.

d. Visualise a handful of preprocessed (ω, ε) maps across the range and confirm that the features evolve smoothly with λ_{SOC}.

e. Estimate the wall-clock time per ED forward solve and plan the training-set size accordingly (the forward solve is the bottleneck).

3. **Training a neural-network regressor for λ_{SOC}.** Train a fully-connected feed-forward neural network that inverts the spectral fingerprint,

$$f_\theta : \mathrm{d}I/\mathrm{d}V(\omega_q, \varepsilon_p) \longmapsto \lambda_{\text{SOC}}, \tag{4}$$

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 and the `MODEL_WIDTHS['soc']` setting in `src/config.py`. 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 λ_{SOC} and the Pearson-correlation fidelity

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

4. **Noise retraining and diagnostics.**

a. **Additive spectral 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 preprocessed $\mathrm{d}I/\mathrm{d}V$ input,

$$\mathrm{d}I/\mathrm{d}V_{\text{noisy}} = \mathrm{d}I/\mathrm{d}V + \mathcal{N}(0, W^2), \tag{6}$$

at a fixed training level (e.g. $W = 0.05$).

b. **Fidelity-versus-noise curve.** For both the clean-trained and noise-retrained models, sweep the test-time noise level $W \in [0, 1]$ (in units of the typical signal amplitude) and plot $\mathcal{F}_{\lambda_{\text{SOC}}}(W)$. Reproduce qualitatively the monotonic decline of Fig. 3(e) of the paper and comment on the robustness gain from noise retraining.

c. **Out-of-distribution testing.** Evaluate the model on λ_{SOC} values just outside the training range and on spectra generated with slightly shifted nuisance parameters (a different τ, a different $[\varepsilon_{\min}, \varepsilon_{\max}]$ window, a different broadening η). Comment on the extrapolation behaviour and on the identifiability of λ_{SOC} when the nuisance parameters are not perfectly known — this is a direct motivation for the full four-parameter problem of the paper.

5. **Performance benchmarking.**

a. Measure the wall-clock time of a single ED forward solve (dominated by the many-body diagonalisation and the three dynamical-correlator evaluations) as a function of the broadening η and the frequency grid N_ω.

b. Report the final fidelity $\mathcal{F}_{\lambda_{\text{SOC}}}$ 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 the full four-parameter set $(\tau, \lambda_{\mathrm{SOC}}, \varepsilon_{\min}, \varepsilon_{\max})$ of the paper, and how the diagnostics of step 4 would need to be extended to a multi-target setting (e.g. separate regressors per parameter, parameter-wise fidelity curves).

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 (molecular-Hamiltonian dataset generation → NN training → noise retraining → diagnostics) for a minimal number of orbitals.

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`, `pandas`, `scikit-learn`, `matplotlib`, TensorFlow / Keras (as in the companion repository; or PyTorch equivalents), DMRGPy (many-body ED / DMRG and dynamical correlators).

References

- G. Lupi et al., arXiv:2601.19371 (2026).
- G. Lupi et al., Molecular-Hamiltonian-Learning — companion GitHub repository (forward solver in `src/helper.py` and `src/simulator.py`; training pipeline in `src/train.py`; configuration in `src/config.py`).
- J. L. Lado, DMRGPy — Python library for DMRG simulations of quasi-1D spin and fermionic systems.