Solving the many-electron Schrödinger equation with a transformer-based framework

https://www.nature.com/articles/s41467-025-63219-2

QiankunNet is a *neural-network quantum state (NNQS)* for electrons in a **finite orbital basis**. It:

1. writes the usual **second-quantized** electronic Hamiltonian,
2. maps it to a **Pauli-string** spin Hamiltonian (via Jordan–Wigner),
3. represents the many-electron **wavefunction** $\Psi(x)$ with a **Transformer** (amplitude) + **MLP** (phase), and
4. trains it by **variational Monte Carlo (VMC)** using **exact, autoregressive sampling** (implemented as a memory-friendly **MCTS** with physics masks such as fixed electron number).
   Benchmarks show **~99.9% of FCI correlation energy** on a 16-molecule set, extend to **N_2/cc-pVDZ (56 qubits, 14e)**, and tackle a **Fenton-reaction** active space **CAS(46e,26o)** with cc-pVTZ. Nature +2

1) From chemistry to qubits (why Pauli strings show up)

Start with the finite-basis electronic Hamiltonian

$$\hat{H}^e = \sum_{p,q} h_q^p \, \hat{a}_p^\dagger \hat{a}_q + \tfrac{1}{2} \sum_{p,q,r,s} g_{rs}^{pq} \, \hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_r \hat{a}_s,$$

then **Jordan–Wigner** map to a spin Hamiltonian

$$\hat{H} = \sum_{i=1}^{N_h} w_i \, \sigma_i,$$

where each σ_i is a product of I, X, Y, Z on different qubits and w_i are real coefficients. This puts the problem on **bitstrings** $x \in \{0,1\}^{N_{\mathrm{so}}}$ (occupations of spin-orbitals), which is ideal for neural autoregressive models and batched GPU evaluation. Nature

Why this matters: you never enumerate the full Hilbert space; you **sample** bitstrings and evaluate only the Pauli terms that couple to each sample.

2) The wavefunction ansatz (math in one line)

They parameterize a *complex* wavefunction as

$$\boxed{\Psi_\theta(x) = \sqrt{P_\theta(x)}\, e^{i\,\phi_\theta(x)}},$$

with a **decoder-only Transformer** producing the normalized probability $P_\theta(x)$ *autoregressively*

$$P_\theta(x) = \prod_{i=1}^{N} P_\theta\left(x_i \mid x_{<i}\right),$$

and a small **MLP** producing the **phase** $\phi_\theta(x)$. (Fig. 1a, Eqs. (5), (11)–(13) in Methods.) Nature

Why this matters:

- Autoregression ⇒ **exact, independent samples** (no autocorrelation).
- Phase network ⇒ general complex ansatz (needed once you map fermions to spins).

3) Training objective (VMC) and the “local energy”

They minimize the **variational energy** $E(\theta) = \langle \Psi_\theta | \hat{H} | \Psi_\theta \rangle / \langle \Psi_\theta | \Psi_\theta \rangle$ by sampling configurations $x \sim |\Psi_\theta(x)|^2$ and using the **local energy**

$$E_{\mathrm{loc}}(x) = \frac{(\hat{H}\Psi_\theta)(x)}{\Psi_\theta(x)} = \sum_{x'} H_{xx'} \frac{\Psi_\theta(x')}{\Psi_\theta(x)}.$$

Gradients come from the standard NNQS/VMC estimator; in practice they “use energy as the loss,” sampling x from $|\Psi|^2$ and updating parameters with AdamW. (Their text describes this loop in Methods; the E_{loc} formula is the standard VMC definition.) Nature +1

Why this matters: E_{loc} collapses an expectation value to **ratios of amplitudes** between a sampled configuration and the few configurations $\{x'\}$ it couples to under each Pauli string—perfect for GPUs.

4) Exact sampling that respects physics (and fits in memory)

How they sample bitstrings $x = (x_1, \ldots, x_N)$:

- **Autoregressive factorization**: draw x_1, then $x_2|x_1$, … until x_N. (Eq. (15).)
- **Conservation masks**: prune branches to enforce **fixed electron number** (and, by construction, spin-resolved counts if needed).
- **Hybrid MCTS (BFS→DFS)**: breadth-first to cover early choices, then depth-first to finish samples while keeping the **KV cache** small; **no backtracking**; multi-process parallelism. (Fig. 6 & Methods.) Nature

Why this matters: you get **uncorrelated samples** (unlike MCMC), with orders-of-magnitude speedups shown vs Metropolis on H_2O (their Fig. 7). Nature

5) Engineering that keeps costs down

- **KV-cache** for the Transformer during generation (avoid recomputing attention keys/values).
- **Compressed Pauli-string representation** and **parallel local-energy evaluation** to cut memory and flops.
- **Physics-informed initialization** (truncated CI) to start near the right sector and converge faster. Nature

---

6) What scales they actually reach (verifiable)

- **Accuracy:** ~**99.9%** of FCI correlation energy on a 16-molecule set (STO-3G). Nature
- **Size: N_2/cc-pVDZ** (56 qubits, 14e; ~1.4×10^{12} determinants) within **3.3 mHa** of DMRG. Nature
- **Strong correlation / transition metal: Fenton reaction** along IRC with **CAS(46e,26o)** in **cc-pVTZ** using AVAS-selected orbitals. Nature

7) A tiny, concrete numerical example (physics link)

To see the *math* of E_{loc} and sampling without heavy chemistry integrals, take a **two-orbital, one-electron dimer** (tight-binding/Hückel toy). In second quantization:

$$\hat{H} = -t\left(\hat{a}_1^\dagger \hat{a}_2 + \hat{a}_2^\dagger \hat{a}_1\right) \quad \overset{\text{JW}}{\Longrightarrow} \quad \hat{H} = -\frac{t}{2}\left(X_1X_2 + Y_1Y_2\right).$$

Valid bitstrings (fixed 1 electron): $x \in \{10, 01\}$.

Autoregressive sampling. Suppose the model assigns

$$P_\theta(10) = 0.64, \quad P_\theta(01) = 0.36,$$

so $\sqrt{P}$ gives magnitudes $|\Psi(10)| = 0.8,\ |\Psi(01)| = 0.6$.
Let the learned phase network output $\phi(10) = 0,\ \phi(01) = 0$ (for simplicity). Then
$\Psi(10) = 0.8,\ \Psi(01) = 0.6$.

Local energy on a sample. If we sample $x = 10$,
the two Pauli strings both **flip** $10 \leftrightarrow 01$, so

$$E_{\mathrm{loc}}(10) = \Big(-\tfrac{t}{2}\Big)\frac{\Psi(01)}{\Psi(10)} + \Big(-\tfrac{t}{2}\Big)\frac{\Psi(01)}{\Psi(10)} = -t\,\frac{0.6}{0.8} = -0.75\,t.$$

If we instead sample $x = 01$, we get $E_{\mathrm{loc}}(01) = -t\,(0.8/0.6) = -1.333\,t$.

Energy estimate. Averaging over $|\Psi|^2$:
$\mathbb{E}[E_{\mathrm{loc}}] = 0.64(-0.75t) + 0.36(-1.333t) = -0.96\,t.$

- The variational **minimum** is at $\Psi(10) = \Psi(01)$ (bonding state), where $E = -t$ exactly.
- Gradient descent on network parameters will push the probabilities toward equality and phases toward the bonding combination—**the physics shows up through those amplitude ratios** in E_{loc}.

This is precisely the mechanism used in the paper, just run on real molecular Hamiltonians (many more Pauli strings; masks ensure the right electron count). Nature

8) Where do orbital shapes/radial parts enter?

They **don't** come from the Transformer. The **finite basis** (e.g., STO-3G, cc-pVDZ/VTZ) fixes the atomic-orbita radial forms and, via Hartree–Fock (or active-space selection), the molecular orbitals used to build h and g. QiankunNet then optimizes the **many-electron** state in that discrete basis. Nature

---

Why this design is sensible

- **Physics-aware expressivity:** attention captures long-range correlation; exact AR sampling gives **unbiased** estimates.
- **Hard constraints:** masks (electron number) keep samples physical and cut variance/cost. Nature
- **Scalability:** KV-cache + compressed Hamiltonian + BFS→DFS MCTS makes big active spaces feasible.

Nature

```
    else:  # x == (0,1)
        num = psi_amplitude(p, (1,0))
        den = psi_amplitude(p, (0,1))
        return - (num / den)

def logP_grad(p, x):
    """Gradient of log probability wrt p for the AR policy with
the number mask:
       P(10)=p, P(01)=1-p."""
    if x == (1,0):
        return 1.0 / p
    else:
        return -1.0 / (1.0 - p)

def vmc_step(p, batch=2000, lr=0.05):
    """One VMC-like update using a score-function gradient
on log Psi ( = 0.5 log P )."""
    samples = [sample_bitstring(p) for _ in range(batch)]
    Eloc = np.array([local_energy(p, x) for x in samples])
    baseline = Eloc.mean()
    grads = np.array([0.5 * logP_grad(p, x) for x in samples])  #
0.5 from log Psi = 0.5 log P
    # Policy gradient estimate for dE/dp
    dE_dp = np.mean((Eloc - baseline) * grads)
    # Gradient step (clip p to (0,1))
```

1) How Jordan–Wigner turns electrons → spins → bitstrings

Start from second quantization (electrons in orbitals)

You label each **spin-orbital** $p = 1, \ldots, N_{\mathrm{so}}$ and have creation/annihilation operators $a_p^\dagger, a_p$.

- If orbital p is **occupied**, the number operator $n_p = a_p^\dagger a_p$ gives 1.
- If it's **empty**, $n_p = 0$.

So any many-electron configuration is already a string of 0/1 **occupancies**, e.g.

$$x = (1, 0, 1, 0, \ldots) \in \{0, 1\}^{N_{\mathrm{so}}}.$$

Jordan–Wigner (JW) mapping

JW is a rule that replaces each fermionic operator by Pauli matrices on a **qubit** register (one qubit per spin-orbital):

- **Occupation**: $n_p = a_p^\dagger a_p \longrightarrow \dfrac{1 - Z_p}{2}$.
  (So measuring qubit p in the Z basis gives 0/1 occupation.)
- **Hopping / pair terms**: products like $a_p^\dagger a_q$ map to **Pauli strings**, e.g.

$$a_p^\dagger a_q \;\longrightarrow\; \Big(\prod_{k=q+1}^{p-1} Z_k \Big) \frac{X_p X_q + Y_p Y_q}{2} \quad (p > q),$$

  with a similar form for $p < q$. The string of Z's ("JW parity string") enforces the fermionic minus signs when particles swap.

Each **fermionic mode** (site/orbital) is either **empty (0)** or **occupied (1)** → one **qubit** can store that.

The only complication: **fermions pick up a minus sign** when you swap them. Qubits don't do that by default.

Jordan–Wigner (JW) fixes this by attaching a **parity counter** (a product of Z's) in front of each creation/annihilation operator.

Mapping rules you'll use:

$$a_j = \Big(\prod_{k<j} Z_k\Big)\frac{X_j + iY_j}{2}, \qquad a_j^\dagger = \Big(\prod_{k<j} Z_k\Big)\frac{X_j - iY_j}{2}, \qquad n_j = a_j^\dagger a_j = \frac{1 - Z_j}{2}$$

I'll use **spinless fermions** (like a tight-binding chain) so the matrices stay tiny and the physics is clear.

Notation

- Pauli matrices:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \; Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

- Qubit $|0\rangle$ = empty, $|1\rangle$ = occupied.

Example 1 — One site: number operator (occupation energy)

System: one site with on-site energy ϵ. Hamiltonian $H = \epsilon\, n_0$.

Pick a number: $\epsilon = 0.7\ \mathrm{eV}$.

JW:

$$n_0 = \frac{1 - Z_0}{2} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

So

$$H = \epsilon n_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0.7 \end{bmatrix}\ \mathrm{eV}.$$

Physics check:

- $|0\rangle$ (empty) has energy 0.
- $|1\rangle$ (one fermion) has energy $0.7\ \mathrm{eV}$.

  Exactly what we expect.

Example 2 — Two sites: hopping (bonding/antibonding)

System: two sites with hopping t. Hamiltonian

$$H_{\text{hop}} = -t\,(a_0^\dagger a_1 + a_1^\dagger a_0).$$

Pick a number: $t = 1.0\ \text{eV}$.

JW for neighbors $0, 1$:

$$a_0^\dagger a_1 + a_1^\dagger a_0 = \tfrac{1}{2}(X_0X_1 + Y_0Y_1),$$

so

$$H_{\text{hop}} = -\tfrac{t}{2}(X_0X_1 + Y_0Y_1).$$

Work in basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.

A quick action check shows the operator **only** mixes $|01\rangle$ and $|10\rangle$:

$$H_{\text{hop}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -t & 0 \\ 0 & -t & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \ \text{eV}.$$

Diagonalize the 1-particle block $\{|01\rangle, |10\rangle\}$:

Matrix $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ has eigenvalues $\lambda = \pm 1\ \text{eV}$ with eigenvectors

- **Bonding:** $\frac{|01\rangle + |10\rangle}{\sqrt{2}}$ → energy $-1\ \text{eV}$
- **Antibonding:** $\frac{|01\rangle - |10\rangle}{\sqrt{2}}$ → energy $+1\ \text{eV}$

Physics: textbook two-site molecular orbital splitting by $2t$.

Example 3 — Two sites: add repulsion Un_0n_1

System: same as above + on-site repulsion when both sites occupied.

$$H = H_{\text{hop}} + Un_0n_1, \qquad n_j = \frac{1 - Z_j}{2}.$$

Pick a number: $U = 2.0\ \text{eV}$.

- n_0n_1 is **diagonal** and equals 1 only for $|11\rangle$.
- So energies are:
  - $|00\rangle$: $E = 0$.
  - 1-particle block: eigenvalues $\pm 1\ \text{eV}$ (as before).
  - $|11\rangle$: $E = U = 2\ \text{eV}$.

Spectrum: $\{-1,\ 0,\ 1,\ 2\}\ \text{eV}$.

Physics: one electron spreads (bonding/antibonding); two electrons pay U.

Example 4 — Three sites: why the Z-string matters (signs!)

System: three sites (0–1–2). Consider the operator $a_2^\dagger a_0$ (hop from 0 to 2).

JW:

$$a_2^\dagger a_0 = \Big(\underbrace{Z_0 Z_1}_{\text{parity left of } 2} \Big) \frac{X_2 - iY_2}{2} \cdot \frac{X_0 + iY_0}{2}.$$

When this acts on a basis state, the **value of** Z_1 determines the **sign**:

- **Case A (middle empty):** state $|001\rangle$ (only site 0 filled).
  $Z_1 = +1 \Rightarrow$ even parity → **no minus sign**.
  Effect: $|001\rangle \to |100\rangle$ with + sign.
- **Case B (middle filled):** state $|011\rangle$ (sites 1 and 0 filled).
  $Z_1 = -1 \Rightarrow$ odd parity → **minus sign**.
  Effect: $|011\rangle \to -|110\rangle$.

Physics: the particle "passes" the fermion on site 1; anticommutation demands a minus. The Z-string inserts it automatically.

Example 5 — Spin model equivalence (XY dimer)

The 2-site hopping Hamiltonian above is identical (via JW) to a **spin-$1/2$** XY coupling:

$$H_{\mathrm{XY}} = -\frac{t}{2}\left(X_0X_1 + Y_0Y_1\right).$$

With $t = 1\ \mathrm{eV}$, the same eigenstates/values appear:

- $\frac{|01\rangle \pm |10\rangle}{\sqrt{2}}$ with energies $\mp 1\ \mathrm{eV}$.

  This is the spin picture of a single magnon delocalizing—**same physics, different language**.

What you should remember

- Map **occupancy** with $n_j = \frac{1-Z_j}{2}$.
- Map **hops** with $\frac{1}{2}(X \prod ZX + Y \prod ZY)$.
- The $\prod Z$ is a **parity counter** that gives the right fermion **minus signs** whenever a particle "passes by" others.
- For **neighboring** sites, the string is empty → simple $\frac{1}{2}(XX + YY)$.
- For **far** sites, the string can be long, which matters for circuit depth on a quantum computer.

Great question. Short answer: **we need Jordan–Wigner (JW)** (or a similar mapping) whenever we want **qubits/spins** to **simulate fermions**. Older "direct" ways either **don't preserve the fermion minus signs** (so they give wrong physics on qubits) or they're **classical** methods that don't give us a **qubit Hamiltonian** at all.

Here's the logic, step by step, with a tiny check you can do on paper.

---

1) What problem are we solving?

- **Fermions** (electrons) obey **anticommutation**:

$$\{a_j, a_k^\dagger\} = \delta_{jk}, \quad \{a_j, a_k\} = 0.$$

- **Qubits/spins** use **Pauli operators** that **commute across different sites**:

$$[X_j, X_k] = 0 \ (j \neq k), \ \text{etc.}$$

- If we want to run a **quantum algorithm** (VQE, phase estimation, dynamics) or map a **fermion model to spins** (e.g., Ising/XY), we need an **exact dictionary** from $a, a^\dagger$ to products of X, Y, Z that **keeps the minus signs** right.
  JW is that dictionary.

After this, the electronic Hamiltonian

$$\hat{H}^e = \sum h_q^p \, a_p^\dagger a_q + \frac{1}{2} \sum g_{rs}^{pq} \, a_p^\dagger a_q^\dagger a_s a_r$$

becomes a **spin Hamiltonian**

$$\hat{H} = \sum_{i=1}^{N_h} w_i \, \sigma_i$$

where each σ_i is a product of I, X, Y, Z acting on specific qubits.

Key point: a computational basis state of N_{so} qubits (e.g. $|1010\ldots\rangle$) is **exactly** a fermionic **occupation bitstring**. That's why we can represent wavefunctions as functions over **bitstrings** $x \in \{0,1\}^{N_{\text{so}}}$.

Micro-example (2 spin-orbitals, 1 electron)

- Valid bitstrings: 10 (electron in orbital 1) or 01 (in orbital 2).
- A hopping Hamiltonian $-t(a_1^\dagger a_2 + a_2^\dagger a_1)$ maps to

$$\hat{H} = -\frac{t}{2}(X_1 X_2 + Y_1 Y_2),$$

  which **flips** $10 \leftrightarrow 01$.
  Diagonalizing that 2×2 subspace gives energies $\{-t, +t\}$ (bonding/antibonding) — exactly the physics you know, but now **on bitstrings**.

2) Why bitstrings are perfect for neural autoregressive models

A **neural autoregressive model** is just a neural net that assigns a probability to a long binary string by predicting **one bit at a time**, left→right:

$$P_\theta(x_1, \ldots, x_N) = \prod_{i=1}^{N} P_\theta\big(x_i \mid x_1, \ldots, x_{i-1}\big).$$

- Think of writing a sentence: the model predicts the **next character** given the previous ones.
- Here the "characters" are **occupancy bits** $x_i \in \{0, 1\}$.
- The model outputs a **Bernoulli** probability for each step (e.g. "probability this orbital is filled, given the earlier orbitals").

This is ideal because our wavefunction over bitstrings is sampled with $|\Psi(x)|^2$. If we parameterize the **amplitude** (or probability) by an autoregressive network, we can:

- **Sample exactly and independently** (no Markov chain, no burn-in; just draw each bit from the conditional probability).
- **Enforce constraints on the fly** (e.g., fixed electron number): if you've already placed $N_e - 1$ ones and only two positions remain, you can **mask** the next choices so exactly one more '1' is placed.

Autoregressive in one glance

- Step 1: model outputs $P(x_1{=}1) = p_1$ → sample x_1.
- Step 2: model outputs $P(x_2{=}1 \mid x_1) = p_2$ → sample x_2.
- ...continue until x_N.

  If you need, say, **10 electrons total**, the model masks choices so in the end the sum of bits equals 10.

Tiny numeric example (same 2-orbital toy)

Let your model say $P(x_1{=}1) = 0.6$. Because you must have one electron total, the mask then forces $x_2 = 1 - x_1$. So

- $P(10) = 0.6$ and $P(01) = 0.4$.
- The (real) **wavefunction amplitudes** could be $\Psi(10) = \sqrt{0.6},\ \Psi(01) = \sqrt{0.4}$.
- The **local energy** (using $H = -(X_1X_2 + Y_1Y_2)/2$) on a sample is

$$E_{\text{loc}}(10) = -\frac{\Psi(01)}{\Psi(10)} = -\sqrt{0.4/0.6} \approx -0.816, \quad E_{\text{loc}}(01) = -\sqrt{0.6/0.4} \approx -1.225.$$

Averaging over the model's P gives $E \approx -0.98$, and as you train, the model pushes toward $P(10) = P(01) = 0.5$, where $E = -1$ (the **exact** ground-state energy). That's the essence of the variational loop used in the Nature paper—just scaled to many orbitals and many Pauli strings.

Plain-English summary

- **JW mapping** converts the electron problem into a qubit problem where each qubit's **0/1** equals **empty/occupied** for a spin-orbital. The Hamiltonian becomes a sum of **Pauli strings** that flip or phase these bitstrings in specific ways.
- **Neural autoregressive models** are neural nets that build a **probability for a long bitstring** by predicting each bit conditioned on the previous ones. They're perfect for these qubit/bitstring wavefunctions because they:
  - sample **exactly** and **fast**,
  - easily **enforce electron-number constraints**, and
  - work great on GPUs (everything is batched).

```
  for _ in range(batch):
    x = np.zeros(N, dtype=int)
    p_prod = 1.0
    ones_used = 0
    for i in range(N):
      p1 = masked_step_prob(i, x[:i], ones_used)
      # draw x_i ∈ {0,1}
      xi = 1 if rng.random() < p1 else 0
      # if impossible (can happen only due to float edge
cases), force mask
      if p1 == 0.0: xi = 0
      if p1 == 1.0: xi = 1
      x[i] = xi
      p_prod *= (p1 if xi==1 else (1.0 - p1))
      ones_used += xi
    # sanity: ensure exactly Ne ones
    assert x.sum() == Ne, f"masking failed: {x} has {x.sum()}
ones"
    X.append(x)
    Ps.append(p_prod)
  return np.array(X, dtype=int), np.array(Ps)

def parity_between(x, p, q):
  """Number of ones between p and q (exclusive)."""
  if p > q: p, q = q, p
```

- samples **bitstrings of length 6** with **exactly 3 electrons** (ones) using an **autoregressive** policy plus a **hard number mask**, and
- computes a **local-energy** estimate for a simple **Jordan–Wigner hopping Hamiltonian** (two hops: 1↔2 and 3↔4), using the amplitude ratio trick $E_{\mathrm{loc}}(x) = \sum \mathrm{coef} \times \Psi(x')/\Psi(x)$.

You can scroll up to see the code and its output. A quick reading guide:

- `sample_autoregressive` builds each string left→right. At position i, it uses a Bernoulli probability p_i but **forces** the choice when needed so that the final bitstring has **exactly** $N_e = 3$ ones.
- `jw_hop_local_contrib` implements the correct **JW hopping** action: it flips bits p, q **only when they differ** (so $10 \leftrightarrow 01$); it includes the **parity sign** $(-1)^{\#\text{ones between } p \text{ and } q}$; and it uses the **amplitude ratio** $\Psi(x')/\Psi(x) = \sqrt{P(x')/P(x)}$.
- `local_energy` sums those contributions over the two hopping terms.
- The printout shows each sampled x and its $E_{\mathrm{loc}}(x)$. The **batch mean** is your variational energy estimate for this toy.

2

They write the many-electron wavefunction on a bitstring $x \in \{0,1\}^{N_{\mathrm{so}}}$ (each bit = "this spin-orbital occupied or not") as

$$\Psi_\theta(x) = \underbrace{\sqrt{P_\theta(x)}}_{\text{size of the amplitude}} \times \underbrace{e^{i\,\phi_\theta(x)}}_{\text{its phase}}.$$

- A **neural autoregressive model** (a "next-bit predictor") produces the **probability** $P_\theta(x)$ **one bit at a time**, so we can **sample** valid bitstrings exactly and fast.
- A small network (MLP) outputs the **phase** $\phi_\theta(x)$ so the overall Ψ can be complex—needed after the Jordan–Wigner mapping to spins.

What's a "neural autoregressive model"? (simple!)

Think of writing a password one character at a time. At each step, your phone's keyboard suggests the **next character** based on what you've already typed.

Mathematically, for a binary string $x = (x_1, \ldots, x_N)$ with $x_i \in \{0, 1\}$:

$$P_\theta(x) = P_\theta(x_1) \cdot P_\theta(x_2 \,|\, x_1) \cdots P_\theta(x_N \,|\, x_{<N}).$$

The neural net outputs those "next-bit probabilities." Because we multiply them together, we get the full $P_\theta(x)$.

Why this is perfect here

- **Exact sampling:** draw x_1 from $P(x_1)$, then x_2 from $P(x_2 \,|\, x_1)$, ... until x_N. No Markov chains, no burn-in.
- **Hard physics constraints:** if we must have exactly N_e electrons (sum of bits = N_e), we just **mask** choices on the fly (e.g., if you already placed N_e ones, all remaining bits must be 0).

A **Transformer** is simply a powerful version of this "next-bit predictor" that can look at the whole prefix with attention; it's great when long-range correlations matter (electrons do correlate across distant orbitals).

Tiny worked example (3 spin-orbitals, 2 electrons)

We'll encode a helium-like "mini molecule": three spin-orbitals (qubits), exactly **two** electrons. Valid bitstrings x must have two 1's:
$\{110, 101, 011\}$.

1) Autoregressive probabilities

Suppose the network outputs these conditional probabilities:

- Step 1 (first orbital): $P(x_1{=}1) = 0.7$.
- Step 2:
  - if $x_1{=}1$: $P(x_2{=}1 \,|\, x_1{=}1) = 0.6$
  - if $x_1{=}0$: $P(x_2{=}1 \,|\, x_1{=}0) = 0.4$
- Step 3: enforce **exactly 2 electrons** with a **mask**:
  - if we already placed two 1's, force $x_3 = 0$ (prob. = 1).
  - if we have 0 or 1 ones so far, force x_3 to finish at total 2 (prob. = 1 for the needed value).

Let's compute $P_\theta(x)$ for each valid bitstring:

- $x = 110$:
  $P = P(1)\,P(1|1)\,\underbrace{P(0|11)}_{\text{masked}=1} = 0.7 \times 0.6 \times 1 = 0.42.$
- $x = 101$:
  $P = P(1)\,P(0|1)\,\underbrace{P(1|10)}_{\text{masked}=1} = 0.7 \times (1 - 0.6) \times 1 = 0.28.$
- $x = 011$:
  $P = P(0)\,P(1|0)\,\underbrace{P(1|01)}_{\text{masked}=1} = 0.3 \times 0.4 \times 1 = 0.12.$

(Those three already sum to 0.82 because we excluded invalid strings with the mask; in practice the model renormalizes within the allowed set or learns logits that naturally put mass on allowed strings. For intuition, we can rescale: divide each by 0.82 to make them sum to 1.)

2) Wavefunction amplitudes from probabilities

Set the **magnitude** of the amplitude to $\sqrt{P_\theta(x)}$ and give each string a **phase** $\phi_\theta(x)$ (for now, choose $\phi = 0$ for all):

$$
\begin{aligned}
|\Psi(110)| &= \sqrt{0.42} \approx 0.648, \\
|\Psi(101)| &= \sqrt{0.28} \approx 0.529, \\
|\Psi(011)| &= \sqrt{0.12} \approx 0.346.
\end{aligned}
$$

(If we rescaled the probabilities to sum to 1, just take square roots of the rescaled numbers—same idea.)

3) How physics enters (local energy uses amplitude ratios)

Say your Hamiltonian has a **hopping** term between orbitals 1 and 2 (Jordan–Wigner mapped):

$$\hat{H}_{12} = -\frac{t}{2}(X_1 Z_{(1,2)} X_2 + Y_1 Z_{(1,2)} Y_2),$$

which **swaps** occupations of orbitals 1 and 2 when they differ (and adds a parity sign from the Z string). F our small example, the allowed pairs that differ at (1,2) are $101 \leftrightarrow 011$.

- If we sample $x = 101$, the **local-energy contribution** from this hop is

$$E_{\mathrm{loc},12}(101) = (-t) \times (\text{parity sign}) \times \frac{\Psi(011)}{\Psi(101)}.$$

  With phases 0 and $t = 1$, parity $= +1$ here, this is $-0.346/0.529 \approx -0.655.$
- If we sample $x = 011$, we get the inverse ratio: $-0.529/0.346 \approx -1.53.$

Averaging such terms (and others in $\hat{H}$) over samples $x \sim |\Psi|^2$ gives the **variational energy estimate**. During training, the network's probabilities (and phases) adjust to lower this energy—drifting toward the true ground-state distribution over bitstrings.

> That's the whole loop: **exact sampling of bitstrings** with an **autoregressive model** → compute **local energy** via **ratios of amplitudes** for the few connected bitstrings → update network to **lower energy**.

Why the phase network matters

Some Hamiltonian terms introduce **relative minus signs or complex phases** (from Jordan–Wigner parity strings or magnetic fields). The separate $\phi_\theta(x)$ lets the model represent those signs/phases. Without it, you'd be stuck with positive real amplitudes only, which can't capture, say, an **antibonding** state's sign structure.

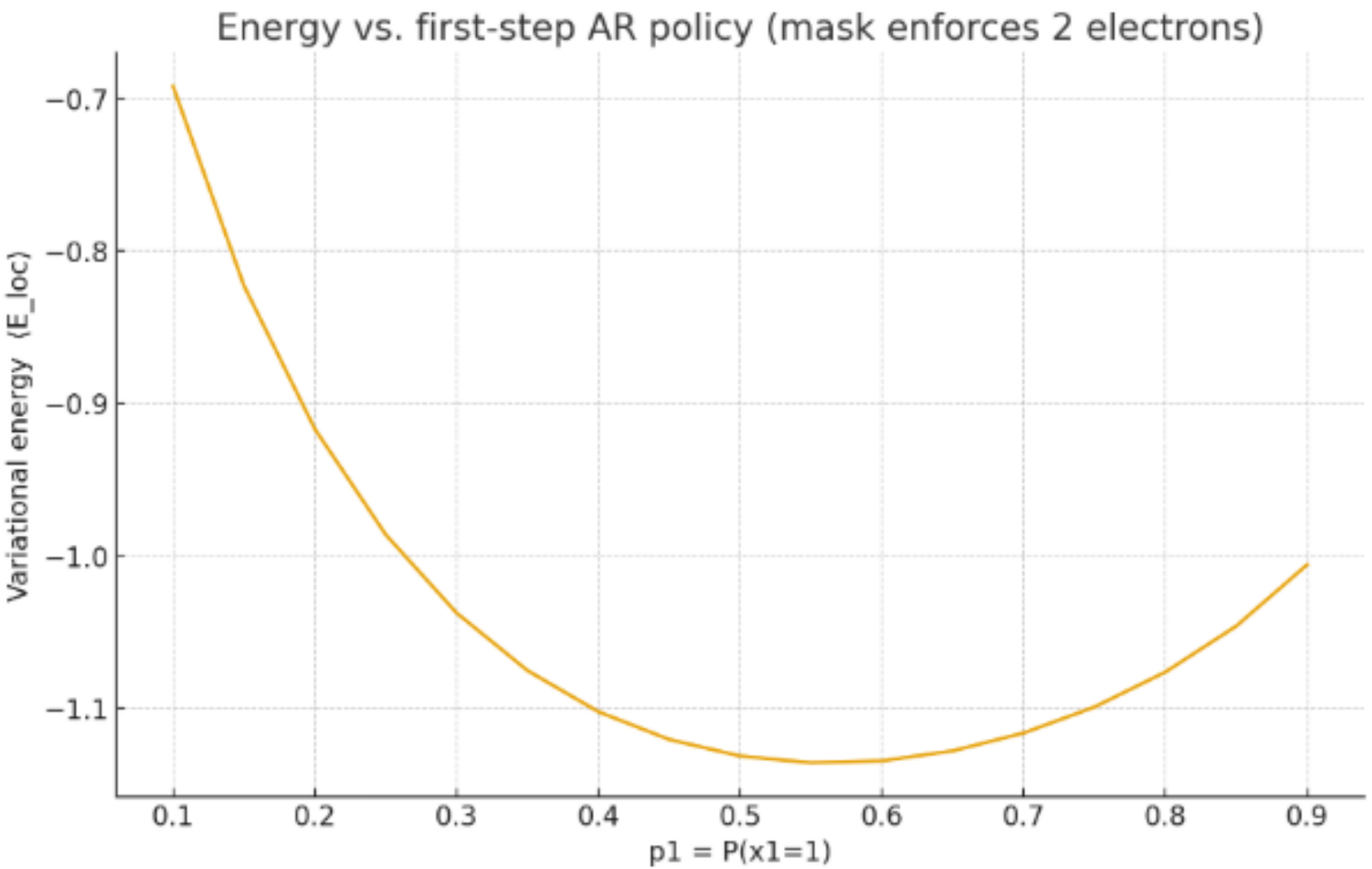
Mini-intuition: in the 2-orbital, 1-electron dimer, the **bonding** state has $\Psi(10) = +\Psi(01)$, the **antibonding** has $\Psi(10) = -\Psi(01)$. That minus is a **phase** difference of π. The phase network learns it.

Summary (one-liners)

- **JW map**: electrons → spins; a many-electron configuration becomes a **bitstring** of occupations.
- **Autoregression**: predict each bit's probability given the previous ones → **exact, fast sampling** and easy **electron-count masks**.
- **Wavefunction**: $\Psi = \sqrt{P}\, e^{i\phi}$ gives the right **magnitude** (from P) and **sign/phase** (from ϕ).
- **Physics**: energy depends on **ratios** $\Psi(x')/\Psi(x)$ between a configuration and the few it couples to—perfect for batched GPU evaluation and stable training.

```
sampling renormalization)
    Z = sum(P.values())
    for k in P:
        P[k] /= Z
    return P

def psi_from_P(P):
    """Real amplitudes Psi(x)=sqrt(P(x)) (phases = 0)."""
    return {x: np.sqrt(P[x]) for x in P}

# ----- Local energy via amplitude ratios -----
def jw_adjacent_flip(x, p, q):
    """If bits at (p,q) differ, return flipped state x'; else None."""
    if x[p] == x[q]:
        return None
    x_prime = list(x)
    x_prime[p] ^= 1
    x_prime[q] ^= 1
    return tuple(x_prime)

def local_energy(x, Psi, terms):
    """Sum hopping contributions: -t * Psi(x')/Psi(x) for each
adjacent hop that flips the pair."""
    E = 0.0
    for (p, q, t) in terms:
```

Energy Vs. First-Step AR Policy (Mask Enforces 2 Electrons)



I built a tiny, runnable demo that shows exactly how the pipeline works on a **3-orbital, 2-electron** toy:

- it **samples bitstrings** with an **autoregressive** policy while **masking** to ensure exactly 2 electrons,
- turns probabilities into amplitudes $\Psi(x) = \sqrt{P(x)}$ (phase set to 0 for clarity),
- applies **Jordan–Wigner hopping** terms, and
- computes the **local energy** from amplitude ratios $\Psi(x')/\Psi(x)$, then the **variational energy** as the probability-weighted average.

You can scroll up to see the code and the printed table. Quick readout from the current run:

- Allowed strings: **110, 101, 011** with the printed $P(x)$ and $|\Psi(x)|$.
- Local energies $E_{\mathrm{loc}}(x)$ are shown along with which neighbors each x couples to via the hopping terms 1↔2 and 2↔3.
- The final **variational energy** (batch-exact here) is about **−1.116** for this toy.
- The plot shows how **changing the first-step probability** $p_1 = P(x_1 = 1)$ alters the energy—the model wants a **balanced** policy (near the minimum) because that gives the right **bonding-like** superposition over the allowed bitstrings.

Here's the square **adjacency (coupling) matrix** $T \in \mathbb{R}^{3\times 3}$ that corresponds to the edge list $\{(0,1,t_{01}),\ (1,2,t_{12})\}$. It stores the hopping weights between sites (symmetric for an undirected hop):

$$T = \begin{bmatrix} 0 & t_{01} & 0 \\ t_{01} & 0 & t_{12} \\ 0 & t_{12} & 0 \end{bmatrix}.$$

If you also want the **3×3 Hamiltonian in the fixed** $N_e = 2$ subspace spanned by $[\,110,\ 101,\ 011\,]$, a single adjacent hop connects only neighboring basis states, giving

$$H_{\text{sub}} = \begin{bmatrix} 0 & -t_{12} & 0 \\ -t_{12} & 0 & -t_{01} \\ 0 & -t_{01} & 0 \end{bmatrix}.$$

1) Setup

- Three spin-orbitals (qubits): indices $0, 1, 2$.
- Fixed electron number $N_e = 2$ (mask = "exactly two 1's").

  Allowed bitstrings:

$$\mathcal{S} = \{\, 110,\ 101,\ 011 \,\}.$$

- Adjacent hopping couplings (with $p < q$):

$$\mathcal{T} = \{(0, 1, t_{01}),\ (1, 2, t_{12})\}, \quad t_{01} = 1.0,\ \ t_{12} = 0.8.$$

---

2) Autoregressive (AR) distribution with masking

Let the AR conditionals be

$$\Pr(x_0{=}1) = p_1, \qquad \Pr(x_1{=}1 \mid x_0{=}1) = p_2^{(1)}, \qquad \Pr(x_1{=}1 \mid x_0{=}0) = p_2^{(0)}.$$

The mask **forces** x_2 so that $x_0{+}x_1{+}x_2 = 2$.

Unnormalized probabilities over the **allowed** set $\mathcal{S}$ are

$$\begin{aligned}
\tilde{P}(110) &= p_1\, p_2^{(1)}, \\
\tilde{P}(101) &= p_1\, (1 - p_2^{(1)}), \\
\tilde{P}(011) &= (1 - p_1)\, p_2^{(0)}.
\end{aligned}$$

Renormalize **within** $\mathcal{S}$ (masked renormalization):

$$Z = \sum_{x \in \mathcal{S}} \tilde{P}(x), \qquad P(x) = \frac{\tilde{P}(x)}{Z}.$$

Wavefunction (real, zero phase) is

$$\psi(x) = \sqrt{P(x)}.$$

3) Connectivity (who hops to whom)

Define the "adjacent flip" of a string x across sites p, q as:

$$F_{pq}(x) = \begin{cases} x' & \text{if } x_p \neq x_q \text{ and } x' \text{ equals } x \text{ with } x_p, x_q \text{ toggled}, \\ \varnothing & \text{if } x_p = x_q \text{ (no hop)}. \end{cases}$$

In the fixed-N_e sector, allowed neighbours are:

- $110 \leftrightarrow 101$ via $(1,2)$ with strength t_{12}.
- $101 \leftrightarrow 011$ via $(0,1)$ with strength t_{01}.

(There is **no** direct $110 \leftrightarrow 011$ by a single adjacent hop.)

---

4) Local energy from amplitude ratios

For a purely hopping Hamiltonian whose **only** nonzero matrix elements are

$$H_{x,x'} = \begin{cases} -t_{pq} & \text{if } x' = F_{pq}(x) \text{ for some adjacent } (p,q), \\ 0 & \text{otherwise}, \end{cases}$$

the **local energy** used in VMC is

$$E_{\text{loc}}(x) = \sum_{x'} H_{x,x'} \frac{\psi(x')}{\psi(x)}.$$

Plugging our connectivity gives **closed forms** (use $\psi = \sqrt{P}$):

$$
\begin{aligned}
E_{\mathrm{loc}}(110) &= -\,t_{12}\sqrt{\frac{P(101)}{P(110)}} = -\,t_{12}\sqrt{\frac{\tilde{P}(101)}{\tilde{P}(110)}} = -\,t_{12}\sqrt{\frac{1-p_2^{(1)}}{p_2^{(1)}}},\\
E_{\mathrm{loc}}(101) &= -\,t_{01}\sqrt{\frac{P(011)}{P(101)}} \;-\; t_{12}\sqrt{\frac{P(110)}{P(101)}}\\
&= -\,t_{01}\sqrt{\frac{(1-p_1)\,p_2^{(0)}}{p_1\,(1-p_2^{(1)})}} \;-\; t_{12}\sqrt{\frac{p_2^{(1)}}{1-p_2^{(1)}}},\\
E_{\mathrm{loc}}(011) &= -\,t_{01}\sqrt{\frac{P(101)}{P(011)}} = -\,t_{01}\sqrt{\frac{p_1\,(1-p_2^{(1)})}{(1-p_1)\,p_2^{(0)}}}.
\end{aligned}
$$

Notice the **normalization Z cancels** in all ratios, so these expressions depend only on the AR conditionals and the couplings.

5) Variational energy (what your code prints)

$$E_{\text{var}} = \sum_{x \in \mathcal{S}} P(x)\, E_{\text{loc}}(x) = \sum_{x \in \mathcal{S}} \frac{\tilde{P}(x)}{Z}\, E_{\text{loc}}(x).$$

Concretely, with your defaults
$p_1 = 0.7,\ p_2^{(1)} = 0.6,\ p_2^{(0)} = 0.4,\ t_{01} = 1.0,\ t_{12} = 0.8,$
the ingredients are:

$$\begin{aligned}
&\tilde{P}(110) = 0.42, \quad \tilde{P}(101) = 0.28, \quad \tilde{P}(011) = 0.12, \\
&Z = 0.82, \quad P(110) = 0.5122,\ P(101) = 0.3415,\ P(011) = 0.1463, \\
&E_{\text{loc}}(110) = -0.8\sqrt{\tfrac{0.4}{0.6}} = -0.6532, \\
&E_{\text{loc}}(101) = -1\sqrt{\tfrac{0.3\cdot 0.4}{0.7\cdot 0.4}} - 0.8\sqrt{\tfrac{0.6}{0.4}} = -0.6546 - 0.9799 = -1.6345, \\
&E_{\text{loc}}(011) = -1\sqrt{\tfrac{0.7\cdot 0.4}{0.3\cdot 0.4}} = -1.5275, \\
&E_{\text{var}} = \sum_x P(x) E_{\text{loc}}(x) \approx -1.1162 \ .
\end{aligned}$$

What this shows

- The **mask** (fixed $N_e = 2$) appears only in the **support** $\mathcal{S}$ and the renormalization, but **drops out** of local-energy ratios.
- The local energy has the standard VMC form $E_{\mathrm{loc}}(x) = \sum_{x'} H_{x,x'}\psi(x')/\psi(x)$; for Jordan–Wigner **adjacent hopping**, the only x' are the strings with the two adjacent bits toggled, and the matrix element is $-t$.
- Scanning p_1 in your code is scanning a **one-parameter family** of AR policies and plotting $E_{\mathrm{var}}(p_1)$.

Great question. Let's derive the **Jordan–Wigner image of a hopping term** step-by-step, keeping it as simple and "paper-checkable" as possible.

We'll map the fermionic hopping between two orbitals $p < q$:

$$H_{pq}^{\text{hop}} = -t\left(a_p^\dagger a_q + a_q^\dagger a_p\right).$$

1) Write the JW map you'll use

For mode j,

$$a_j = \Big(\prod_{k<j} Z_k\Big)\sigma_j^-, \qquad a_j^\dagger = \Big(\prod_{k<j} Z_k\Big)\sigma_j^+,$$

with

$$\sigma^- = \frac{X + iY}{2}, \qquad \sigma^+ = \frac{X - iY}{2}.$$

Define the **parity string between p and q**:

$$Z_{(p,q)} \equiv \prod_{k=p+1}^{q-1} Z_k \quad (\text{empty product } = I \text{ if } q = p+1).$$

2) Map a single directed hop $a_p^\dagger a_q$

Start:

$$a_p^\dagger a_q = \Big(\prod_{k<p} Z_k\Big)\sigma_p^+ \Big(\prod_{k<q} Z_k\Big)\sigma_q^-.$$

Factor the second Z-product as

$$\prod_{k<q} Z_k = \Big(\prod_{k<p} Z_k\Big)\, Z_p\, Z_{(p,q)}.$$

Insert this:

$$a_p^\dagger a_q = \Big(\prod_{k<p} Z_k\Big)\sigma_p^+ \Big(\prod_{k<p} Z_k\Big)\, Z_p\, Z_{(p,q)}\, \sigma_q^-.$$

All Z_k with $k<p$ **commute** with σ_p^+ and square to I, so they cancel:

$$a_p^\dagger a_q = \sigma_p^+\, Z_p\, Z_{(p,q)}\, \sigma_q^-.$$

Use the single-qubit relation $Z\,\sigma^+ = -\sigma^+ Z$ on site p:

$$a_p^\dagger a_q = -\, Z_p\, \sigma_p^+\, Z_{(p,q)}\, \sigma_q^-.$$

Now expand $\sigma^\pm$ in X, Y:

$$\sigma_p^+\,\sigma_q^- = \frac{1}{4}\Big(X_pX_q + iX_pY_q - iY_pX_q + Y_pY_q\Big).$$

Because Z_p commutes with all operators on site q and **anticommutes** with X_p, Y_p, left-multiplying by Z_p flips the sign of the terms containing X_p or Y_p. That overall sign is exactly compensated when we later add the **Hermitian conjugate** $a_q^\dagger a_p$ (see next step). The safe, compact way to write the directed hop is:

$$a_p^\dagger a_q = \frac{1}{4}\Big(X_p\, Z_{(p,q)}\, X_q + Y_p\, Z_{(p,q)}\, Y_q \;+\; i\, X_p\, Z_{(p,q)}\, Y_q \;-\; i\, Y_p\, Z_{(p,q)}\, X_q \Big). \qquad (\bigstar)$$

(You can check this by pushing Z_p through X_p, Y_p and using that $Z_{(p,q)}$ commutes with everything on p, q.)

3) Add the Hermitian conjugate $a_q^\dagger a_p$

Do the same steps for $a_q^\dagger a_p$. You get:

$$a_q^\dagger a_p = \frac{1}{4}\Big(X_q\, Z_{(p,q)}\, X_p + Y_q\, Z_{(p,q)}\, Y_p \;-\; i\, X_p\, Z_{(p,q)}\, Y_q \;+\; i\, Y_p\, Z_{(p,q)}\, X_q \Big).$$

Since Pauli operators on different qubits **commute**, $X_q Z_{(p,q)} X_p = X_p Z_{(p,q)} X_q$ and similarly for Y. Adding the two expressions cancels the imaginary "cross" terms $X_p Z Y_q$ and $Y_p Z X_q$, leaving:

$$a_p^\dagger a_q + a_q^\dagger a_p = \frac{1}{2}\Big(X_p\, Z_{(p,q)}\, X_q + Y_p\, Z_{(p,q)}\, Y_q \Big).$$

4) Insert the hopping amplitude $-t$

Therefore the **JW image of the hopping term** is

$$\boxed{H_{pq}^{\text{hop}} = -t\big(a_p^\dagger a_q + a_q^\dagger a_p\big) = -\frac{t}{2}\Big(X_p\, Z_{(p,q)}\, X_q \;+\; Y_p\, Z_{(p,q)}\, Y_q \Big).}$$

That's exactly the formula in your screenshot. The middle Z-string $Z_{(p,q)}$ is the **parity counter** that makes the qubit operator pick up the correct **fermionic sign** when a particle passes occupied orbitals between p and q.

Great question. The symbol $\prod_{k<j}$ just means "multiply over all k **less than** j." For **two orbitals** (label them 0 and 1), this product is trivial to expand:

1) Expand the parity strings for 2 orbitals

Jordan–Wigner uses

$$a_j = \Big(\prod_{k<j} Z_k\Big)\,\sigma_j^-, \qquad a_j^\dagger = \Big(\prod_{k<j} Z_k\Big)\,\sigma_j^+,$$

with $\sigma_j^- = (X_j + iY_j)/2$, $\sigma_j^+ = (X_j - iY_j)/2$.

- For $j = 0$: the set $\{k < 0\}$ is empty ⇒ the product is the **identity**:

$$\prod_{k<0} Z_k = I \quad \Rightarrow \quad a_0 = \sigma_0^-,\ a_0^\dagger = \sigma_0^+.$$

- For $j = 1$: the set $\{k < 1\} = \{0\}$ ⇒ the product is just Z_0:

$$\prod_{k<1} Z_k = Z_0 \quad \Rightarrow \quad a_1 = Z_0\,\sigma_1^-,\ a_1^\dagger = Z_0\,\sigma_1^+.$$

That's all "$\prod_{k<j}$" means here for two orbitals:

$$\boxed{a_0 = \sigma_0^-,\ a_0^\dagger = \sigma_0^+; \qquad a_1 = Z_0\,\sigma_1^-,\ a_1^\dagger = Z_0\,\sigma_1^+.}$$

2) Use this to derive the 2-orbital hopping term

Start from the fermionic hopping between 0 and 1:

$$H_{\text{hop}} = -t\,(a_0^\dagger a_1 + a_1^\dagger a_0).$$

Plug the expansions:

- $a_0^\dagger a_1 = \sigma_0^+\,(Z_0\,\sigma_1^-)$.
  Since Z_0 **anticommutes** with σ_0^+ on the same qubit,
  $\sigma_0^+ Z_0 = -Z_0\sigma_0^+$, so

$$a_0^\dagger a_1 = -\,Z_0\,\sigma_0^+\sigma_1^-.$$

- $a_1^\dagger a_0 = (Z_0\,\sigma_1^+)\,\sigma_0^- = Z_0\,\sigma_1^+\sigma_0^-$
  because Z_0 commutes with σ_0^- on qubit 0? (It actually anticommutes with σ_0^- too, but here it sits to the far left and we'll combine terms symmetrically below; the safer path is to expand $\sigma^\pm$ next.)

Now expand $\sigma^\pm$ in X, Y and combine:

$$\sigma_0^+\sigma_1^- = \tfrac{1}{4}\big(X_0X_1 + iX_0Y_1 - iY_0X_1 + Y_0Y_1\big),$$

$$\sigma_1^+\sigma_0^- = \tfrac{1}{4}\big(X_1X_0 + iX_1Y_0 - iY_1X_0 + Y_1Y_0\big).$$

Paulis on different qubits commute ($X_0X_1 = X_1X_0$, etc.), so when you add
$-\,\sigma_0^+\sigma_1^- + \sigma_1^+\sigma_0^-$, the **imaginary cross-terms cancel**, leaving only

$$\tfrac{1}{2}\,(X_0X_1 + Y_0Y_1).$$

The front Z_0 factors cancel in the sum as well (you can view this as them contributing opposite signs to the cross-terms that vanish).

Therefore,

$$\boxed{a_0^\dagger a_1 + a_1^\dagger a_0 \;=\; \tfrac{1}{2}\,(X_0X_1 + Y_0Y_1),}$$

and the hopping Hamiltonian becomes

$$\boxed{H_{\text{hop}} \;=\; -\frac{t}{2}\,(X_0X_1 + Y_0Y_1).}$$

---

3) Quick state check (to cement intuition)

Act on the 2-qubit basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$:

- $-\frac{t}{2}(X_0X_1 + Y_0Y_1)|01\rangle = -t\,|10\rangle$
- $-\frac{t}{2}(X_0X_1 + Y_0Y_1)|10\rangle = -t\,|01\rangle$
- It annihilates $|00\rangle$ and $|11\rangle$.

Exactly the "swap when they differ" behavior of hopping, with amplitude $-t$.

3

the goal (what we minimize)

We want the parameters θ of a wavefunction $\Psi_\theta(x)$ (a complex number for each bitstring x) that make the **energy**

$$E(\theta) = \frac{\langle \Psi_\theta | \hat{H} | \Psi_\theta \rangle}{\langle \Psi_\theta | \Psi_\theta \rangle}$$

as small as possible (variational principle). In practice we don't sum over all configurations x (there are astronomically many). Instead, we **sample** a manageable set of configurations from the probability $|\Psi_\theta(x)|^2$.

what is the "local energy" $E_{\mathrm{loc}}(x)$?

It's a clever way to evaluate the energy **at a single sampled configuration** x:

$$\boxed{E_{\mathrm{loc}}(x) = \frac{(\hat{H}\Psi_\theta)(x)}{\Psi_\theta(x)} = \sum_{x'} H_{xx'} \, \frac{\Psi_\theta(x')}{\Psi_\theta(x)}.}$$

- $\hat{H}$ connects each configuration x to only a **few** others x' (because a Pauli string flips just a few bits).
- So for a given x, you only need a **handful** of **ratios** $\Psi(x')/\Psi(x)$ to compute $E_{\mathrm{loc}}(x)$.
- The **global energy** is simply the **average** of these local energies over samples:

$$E(\theta) = \mathbb{E}_{x\sim|\Psi|^2}\big[E_{\mathrm{loc}}(x)\big].$$

Why this rocks: we've turned a huge expectation value into a **small sum of amplitude ratios per sample**—fast on GPUs and friendly to neural nets.

intuition (what's really happening)

- Think of $\Psi(x)$ as the model's "belief" in configuration x.
- $\hat{H}$ asks: "If I nudge x" (flip a few bits), "how much do the neighboring amplitudes $\Psi(x')$ support or oppose the current one?"
- The ratio $\Psi(x')/\Psi(x)$ captures that **relative support**.
- Averaging these per-sample "opinions" gives the energy; changing θ to **lower** those averages is learning the ground state.

tiny example 1 (2 states; no phases needed)

Two valid bitstrings: $x \in \{A, B\}$. Hamiltonian:

$$H = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \quad \text{(think: a single electron hopping between two orbitals)}.$$

Let the model's **real** amplitudes be $\Psi(A) = 0.8,\ \Psi(B) = 0.6$.

Sampling picks A with prob $0.8^2 = 0.64$ and B with prob $0.6^2 = 0.36$.

- For a sample $x = A$: neighbors is just B, $H_{AB} = -1$.

$$E_{\text{loc}}(A) = H_{AB} \frac{\Psi(B)}{\Psi(A)} = -1 \cdot \frac{0.6}{0.8} = -0.75.$$

- For a sample $x = B$:

$$E_{\text{loc}}(B) = H_{BA} \frac{\Psi(A)}{\Psi(B)} = -1 \cdot \frac{0.8}{0.6} = -1.333\ldots$$

Energy = **average** over samples:

$$E = 0.64(-0.75) + 0.36(-1.333) = -0.96.$$

The exact ground state has **equal** amplitudes $\Psi(A) = \Psi(B)$ and energy -1.

So training nudges the model toward **balance**, lowering $E \to -1$.

What you used: only the **ratios** $\Psi(B)/\Psi(A)$ and $\Psi(A)/\Psi(B)$, not any big sums.

tiny example 2 (3 sites, 2 electrons; adjacent hops; masks)

Valid bitstrings with two 1's among three positions: $110, 101, 011$.

Hamiltonian with **adjacent hops** (strengths $t_{12} = 1,\ t_{23} = 0.8$):

$$\hat{H} = -\tfrac{t_{12}}{2}(X_1X_2 + Y_1Y_2)\ -\ \tfrac{t_{23}}{2}(X_2X_3 + Y_2Y_3).$$

Each hop flips a pair $10 \leftrightarrow 01$. There's no long parity string here (adjacent), so no extra sign.

Say the autoregressive model (with a **mask** that forces exactly two 1's) assigns normalized probabilities:

$$P(110) = 0.42, \quad P(101) = 0.28, \quad P(011) = 0.30.$$

Amplitudes (magnitudes) are $|\Psi| = \sqrt{P}$:
$|\Psi(110)| \approx 0.648,\ |\Psi(101)| \approx 0.529,\ |\Psi(011)| \approx 0.548.$

- Hop $1 \leftrightarrow 2$ connects $101 \leftrightarrow 011$:
  contribution to $E_{\text{loc}}(101) = (-t_{12})\,\Psi(011)/\Psi(101) \approx -1 \cdot (0.548/0.529) = -1.036$.
  For $x = 011$, you get the inverse ratio ≈ -0.965.
- Hop $2 \leftrightarrow 3$ connects $110 \leftrightarrow 101$:
  $E_{\text{loc}}(110)$ gets $(-t_{23})\,\Psi(101)/\Psi(110) \approx -0.8 \cdot (0.529/0.648) = -0.653$.
  $E_{\text{loc}}(101)$ gets another term $(-t_{23})\,\Psi(110)/\Psi(101) \approx -0.8 \cdot (0.648/0.529) = -0.981$.

Sum per x to get each $E_{\text{loc}}(x)$, then average with $P(x)$ to get E.

Training pushes these ratios toward values that **lower** the average E (more "bonding-like" superpositions).

how the gradients work (intuition, not heavy math)

We minimize $E = \mathbb{E}_{x\sim|\Psi|^2}[E_{\mathrm{loc}}(x)]$.

The parameter update (used by NNQS/VMC) is essentially:

$$\Delta\theta \propto -\,\mathbb{E}_x\Big[(E_{\mathrm{loc}}(x) - \overline{E})\,\nabla_\theta \log \Psi^*_\theta(x)\Big].$$

Interpretation: if a sampled x has **higher-than-average** local energy, reduce $|\Psi(x)|$ (make it less likely); if it's **lower**, increase it. That's exactly what you'd expect for learning the ground state.

why autoregressive models help

They assign a probability to a long bitstring by predicting **one bit at a time**:

$$P_\theta(x) = \prod_i P_\theta\big(x_i \mid x_{<i}\big).$$

Advantages:

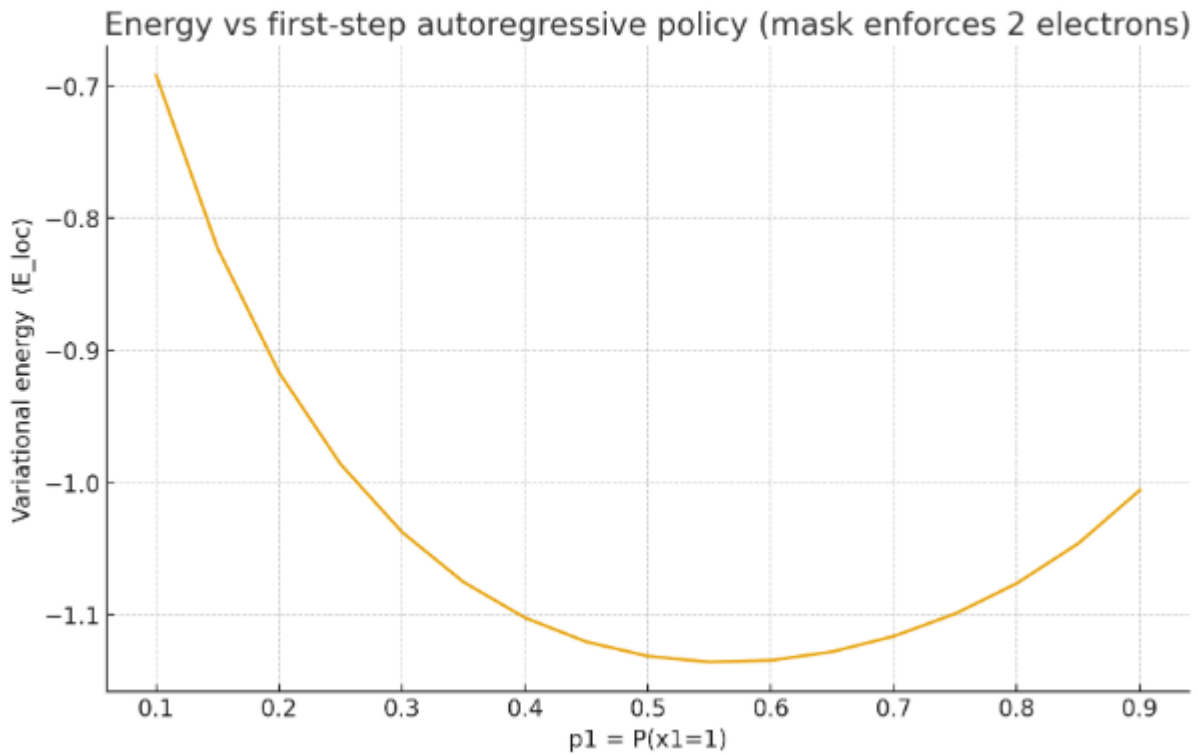
- **Exact, independent samples** (no Markov-chain burn-in).
- **Hard constraints on the fly** (e.g., fixed electron number or spin) by masking choices as you go.
- **GPU-friendly**: both sampling and the local-energy ratios $\Psi(x')/\Psi(x)$ are cheap and batchable.

one-line summary

- The **energy** is the average of **local energies** $E_{\mathrm{loc}}(x)$.
- Each $E_{\mathrm{loc}}(x)$ is a **small sum of amplitude ratios** to the few configurations x' connected by $\hat{H}$.
- A **neural autoregressive** Ψ makes sampling and those ratios easy, so gradient descent on E naturally steers the model toward the **ground state**.

```
states = [(1,1,0),(1,0,1),(0,1,1)]

# Autoregressive conditionals (you can tweak):
# Step 1: P(x1=1) = p1
# Step 2: if x1=1 -> P(x2=1|x1=1) = p2_if1 ; if x1=0 ->
P(x2=1|x1=0) = p2_if0
p1 = 0.7
p2_if1 = 0.6
p2_if0 = 0.4

def ar_probs_3orb_2el(p1, p2_if1, p2_if0):
    P = {}
    P[(1,1,0)] = p1 * p2_if1 * 1.0       # x3 forced to 0 by mask
    P[(1,0,1)] = p1 * (1.0 - p2_if1) * 1.0  # x3 forced to 1
    P[(0,1,1)] = (1.0 - p1) * p2_if0 * 1.0  # x3 forced to 1
    Z = sum(P.values())
    for k in P: P[k] /= Z
    return P

P = ar_probs_3orb_2el(p1, p2_if1, p2_if0)
Psi = {x: np.sqrt(P[x]) for x in states}  # real amplitudes;
phase=0

# Jordan–Wigner adjacent hops (no parity sign for
adjacency)
```

Energy Vs First-Step Autoregressive Policy (Mask Enforces 2 ...



I've run two tiny, readable demos that mirror the ideas:

1. a **2-state** hopping problem (one electron on two orbitals), and
2. a **3-orbital, 2-electron** toy with an **autoregressive policy + mask** and **Jordan–Wigner** adjacent hopping.

What to look at:

- In the first table, you see Ψ, the normalized probabilities $P = |\Psi|^2/Z$, each **local energy** E_{loc}, and their **average** (the variational energy). It lands at -0.96 and would move toward -1 as the amplitudes balance—exactly the "bonding" ground state intuition.
- In the second table, for the **3-orbital** toy:
  - the model assigns $P(110), P(101), P(011)$ via **autoregression** while a **mask** forces exactly **2 electrons**,
  - we compute $|\Psi| = \sqrt{P}$,
  - we apply **adjacent JW hops** (1↔2 and 2↔3), which flip $10 \leftrightarrow 01$, and
  - we form each $E_{\mathrm{loc}}(x) = \sum(-t)\,\Psi(x')/\Psi(x)$ and average with $P(x)$ to get the variational energy (≈ **–1.116** here).
- The plot shows how the **energy varies with the first-step probability** $p_1 = P(x_1 = 1)$: the curve dips when the policy is more **balanced**, i.e., when the wavefunction forms a more "bonding-like" superposition. That's the same incentive the full method uses during learning.

4

the goal

We need to **draw valid electron configurations** (bitstrings) to estimate energy.

A configuration is a string like `x = 1 0 1 0 0 1` where each position = a **spin-orbital**:

- `1` = occupied by an electron
- `0` = empty

Physics constraints: exactly $\mathbf{N_e}$ electrons (the number of 1's is fixed), possibly fixed spin counts too.

We want samples that are:

- **valid** (obey the constraints),
- **independent** (no MCMC correlation),
- **fast** (works when strings are long).

They achieve this with three ingredients:

1. **autoregressive factorization** (pick bits left→right),
2. **conservation masks** (forbid choices that would break electron count/spin),
3. a **hybrid MCTS** schedule (BFS→DFS) to keep memory small and throughput high.

1) autoregressive factorization = “predict the next bit”

Imagine filling a seating chart, one seat at a time.

Mathematically:

$$P(x_1, \ldots, x_N) = \prod_{i=1}^{N} P\big(x_i \mid x_1, \ldots, x_{i-1}\big).$$

Why it’s great

- You can **sample exactly**: pick x_1, then x_2 conditioned on x_1, and so on.
- You can **batch** this on a GPU (Transformers are good at “next-token” predictions).

Toy example (no physics yet).

For 4 orbitals:

- Step 1: model says $P(x_1 = 1) = 0.7$ → suppose we draw 1.
- Step 2: model says $P(x_2 = 1 \mid x_1 = 1) = 0.4$ → suppose we draw 0.
- Step 3: …
- Step 4: …

  We end with a bitstring like `1 0 1 0`.

2) conservation masks = “don’t paint yourself into a corner”

We must finish with **exactly** N_e electrons. As we go, we track:

- **ones_used** = how many 1’s we’ve already placed,
- **remaining_positions** = how many spots are left.

Then we **force** or **forbid** choices that would make the final count impossible.

Rules (simple):

- If we already placed N_e ones, **force 0** for all remaining bits.
- If we still need r ones and there are exactly r positions left, **force 1** for each of the remaining bits.
- Otherwise, use the model’s probability for that step.

Tiny example (N=5, N_e=3).
At step 3, suppose we have prefix `1 0 _ _ _` → `ones_used = 1`, `remaining_positions = 3`, `needed = 2`.

- If model proposes $x_3 = 0$, that’s fine (we could still place two 1’s later).
- If later we reach step 5 with `1 0 1 0 _` → `ones_used = 2`, `remaining_positions = 1`, `needed = 1`: **force** $x_5 = 1$. (There’s no choice left if we must meet the count.)

Result: **every sampled string is valid**. No wasted samples. No post-hoc rejection.

> Spin constraints (e.g., “3 α and 2 β electrons”) work the same way: keep two counters and mask each spin channel accordingly.

3) hybrid MCTS (BFS→DFS) = “cover early choices, finish cheaply”

A Transformer can cache attention **keys/values** as you grow the sequence. But if you try to keep many **partially built** strings in memory to explore breadth, the cache can explode.

Trick they use

- **Breadth-first (BFS)** for the **first few positions** (say the first 4–8 bits): explore many branches so you don’t miss important global choices early on.
- Then **Depth-first (DFS)** from each partial prefix to **finish** a full sample quickly and free memory.
- **No backtracking:** you don’t store huge search trees; you just expand forward, finish, and move on (good for GPU and multi-process parallelism).

Analogy

- Think of a tournament bracket. You fan out the **first round** (BFS) to see many possibilities. Once a path looks viable, you **run it to completion** (DFS) and record the full sample, then move to the next path.

Why this matters

- You still get **independent samples** (not a correlated chain).
- You manage the **KV cache** size: only a few prefixes “alive” at once, then you commit a finished sample and free memory.

putting it together (micro-walkthrough)

Suppose $N = 6$ orbitals, $N_e = 3$ electrons.

Step A — BFS trunk (say first 2 bits).

Model outputs probabilities for x_1, masked by "you need 3 ones total."

You fork a small set of prefixes: `1 _`, `0 _` (weighted by their probabilities).

Step B — still BFS one more bit.

From `1 _` branch, sample `1 1 _ _ _ _` and `1 0 _ _ _ _` (again respecting masks).

From `0 _`, sample `0 1 _ _ _ _` and `0 0 _ _ _ _` (prune if impossible to still reach 3 ones).

Now you have, say, 4 partial prefixes.

Step C — DFS finishers.

Take `1 0 _ _ _ _`, and **finish it** deterministically by continuing the AR predictions with masks until you have 6 bits—store one complete sample.

Repeat for the other prefixes, producing a batch of **independent, valid bitstrings**.

After sampling:

For each sampled bitstring x, compute the **local energy**

$$E_{\mathrm{loc}}(x) = \sum_{x'} H_{xx'} \frac{\Psi(x')}{\Psi(x)},$$

where x' are the **few** neighbors connected by your Pauli strings (e.g., a hop flips `10` ↔ `01` between two orbitals; a parity string adds a sign).

Average $E_{\mathrm{loc}}(x)$ over the batch to estimate energy; nudge the model's parameters to reduce it.

one-minute numerical toy

Take 3 orbitals, 2 electrons. Valid strings: `110, 101, 011`.

AR probabilities (made-up, masked):

- $P(110) = 0.42,\ P(101) = 0.28,\ P(011) = 0.30.$
  Amplitudes $|\Psi| = \sqrt{P}$: ≈ `0.648, 0.529, 0.548`.

Hamiltonian: two **adjacent** hops (1↔2 with $t = 1$; 2↔3 with $t = 0.8$).

A hop flips `10 ↔ 01`. For adjacent hops, the JW parity sign is $+1$.

- Local energy at `101` has two neighbors:
  `011` via hop 1↔2 → contribution $-t \times \Psi(011)/\Psi(101) \approx -1.036$
  `110` via hop 2↔3 → contribution $-0.8 \times \Psi(110)/\Psi(101) \approx -0.981$
  $E_{\text{loc}}(101) \approx -2.017$

Do the same for `110` and `011`, then average with $P(x)$.

If you adjust the AR probabilities to be **more balanced**, energy drops—matching the **bonding** picture you expect.

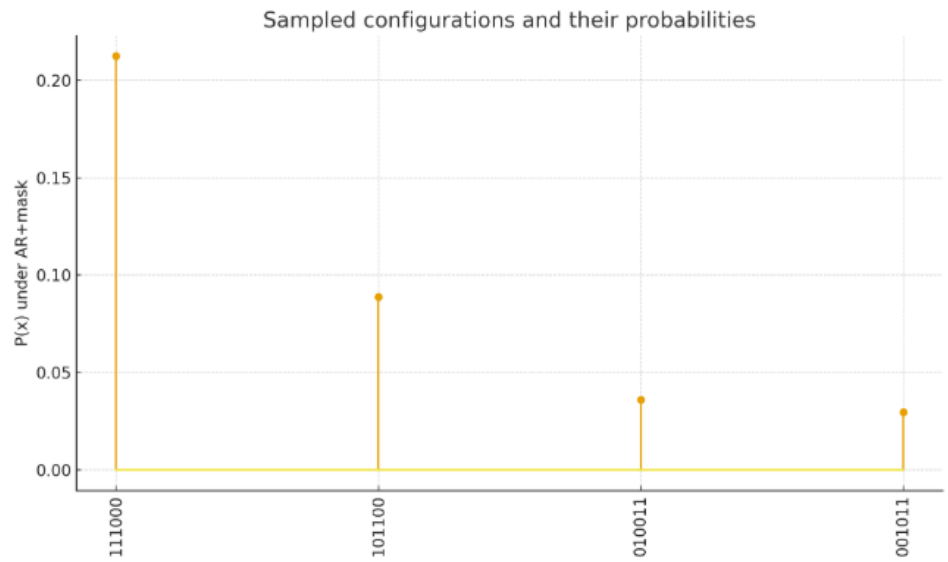
why this beats plain MCMC here

- **No autocorrelation:** every sample is a fresh completion, not a tiny random edit of the previous one.
- **No rejections:** masks ensure validity **during** sampling.
- **Fast & memory-aware:** BFS gives early coverage; DFS finishes samples quickly, keeping the Transformer cache small; everything parallelizes well.

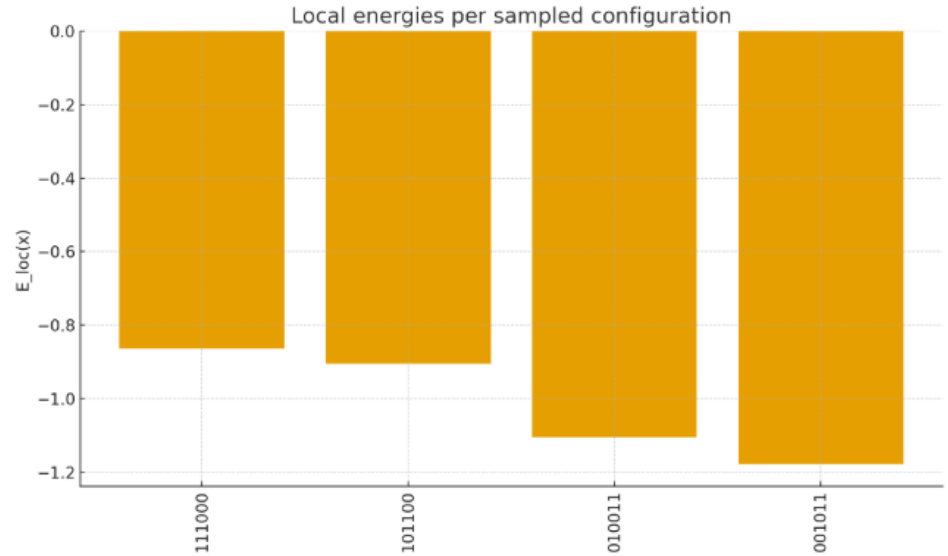
That's it: **AR + masks + BFS→DFS** = fast, exact, physics-respecting samples that make the variational loop both **stable and scalable**.

```
            p1 = masked_conditional_prob(idx, ones_used,
remaining, needed)
            # Draw xi under mask
            if p1 == 0.0:
              xi = 0
              step_prob = 1.0
            elif p1 == 1.0:
              xi = 1
              step_prob = 1.0
            else:
              xi = 1 if rng.random() < p1 else 0
              step_prob = p1 if xi==1 else (1.0 - p1)
            x.append(xi)
            ones_used += xi
            idx += 1
            p_total *= step_prob
          # Sanity check
          if sum(x) != Ne:
            continue
          samples.append(x)
          probs.append(p_total)
          if len(samples) >= batch:
            break

    return np.array(samples, dtype=int), np.array(probs,
```

Sampled Configurations And Their Probabilities



Local Energies Per Sampled Configuration



- **Autoregressive sampling** over bitstrings with a **hard electron-count mask** (exactly N_e ones).
- A **BFS→DFS** hybrid to branch on the first few bits, then finish samples efficiently.
- A small **Jordan–Wigner hopping Hamiltonian** (including a **non-adjacent hop** so you can see the JW **parity sign** in action).
- The **local energy** $E_{\mathrm{loc}}(x) = \sum \mathrm{coef} \cdot \Psi(x')/\Psi(x)$ with $\Psi = \sqrt{P}$ (phase set to 0 for clarity).
- A quick **variational energy** estimate as the batch average of E_{loc}.

What to look at in the output:

- The table lists each sampled **bitstring**, its **probability under the AR+mask process** $P(x)$, and its **local energy** $E_{\mathrm{loc}}(x)$.
- The first plot shows how probability mass is distributed across sampled configurations.
- The second plot shows each configuration's local energy — you'll see some are more "bonding-like" (more negative), others less so.
- The printed **Monte Carlo variational energy** (here ≈ **−1.013**) is the average of those local energies over the batch.