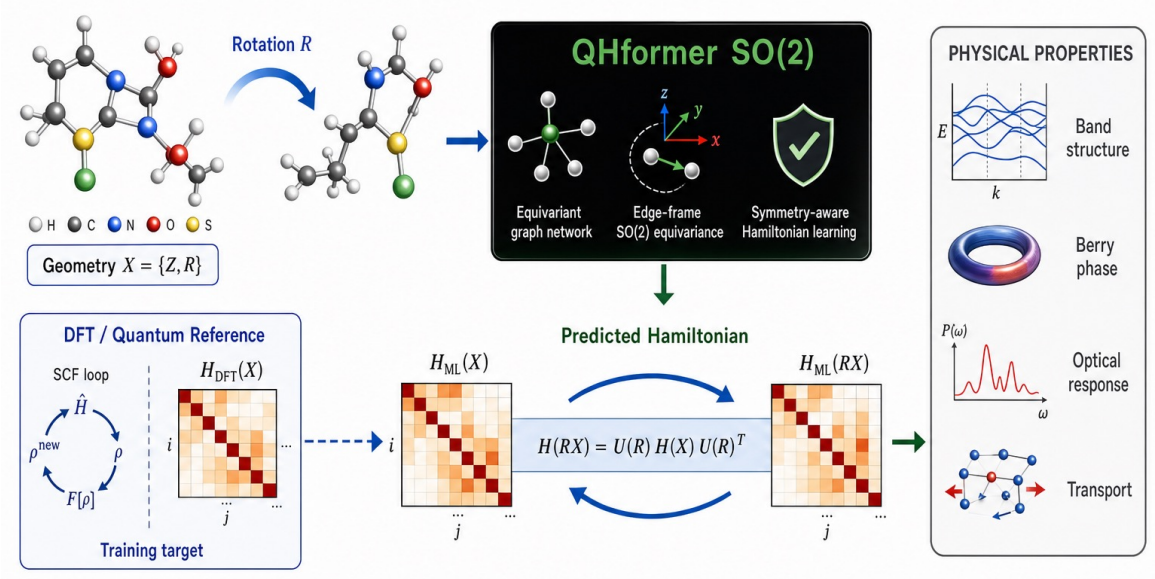
中国科学院大学 前沿交叉科学学院
University of Chinese Academy of Sciences School of Advanced Interdisciplinary Sciences

Full Differentiable TDDFT Code for Neural Network XC Functional, and Attention Based Hamiltonian Learning Model

GradTDDFT
AI-native differentiable TDDFT in JAX

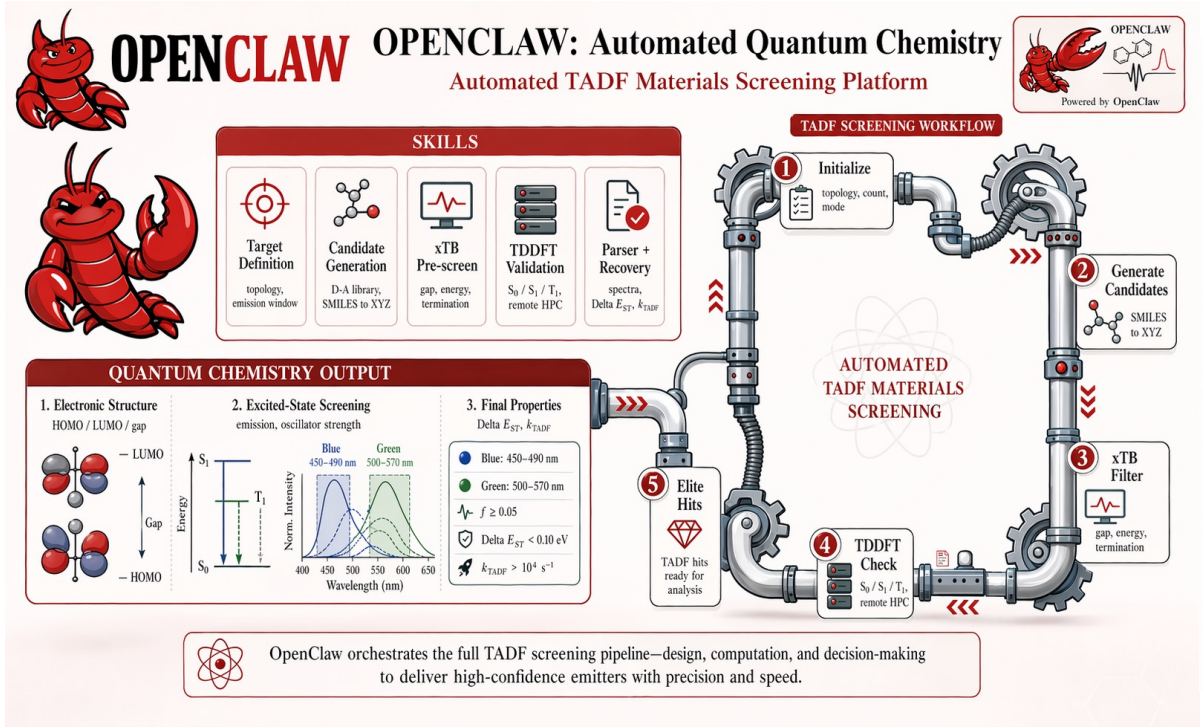
Yuan Jiao
School of Advanced Interdisciplinary Sciences, SAIS
University of Chinese Academy of Sciences

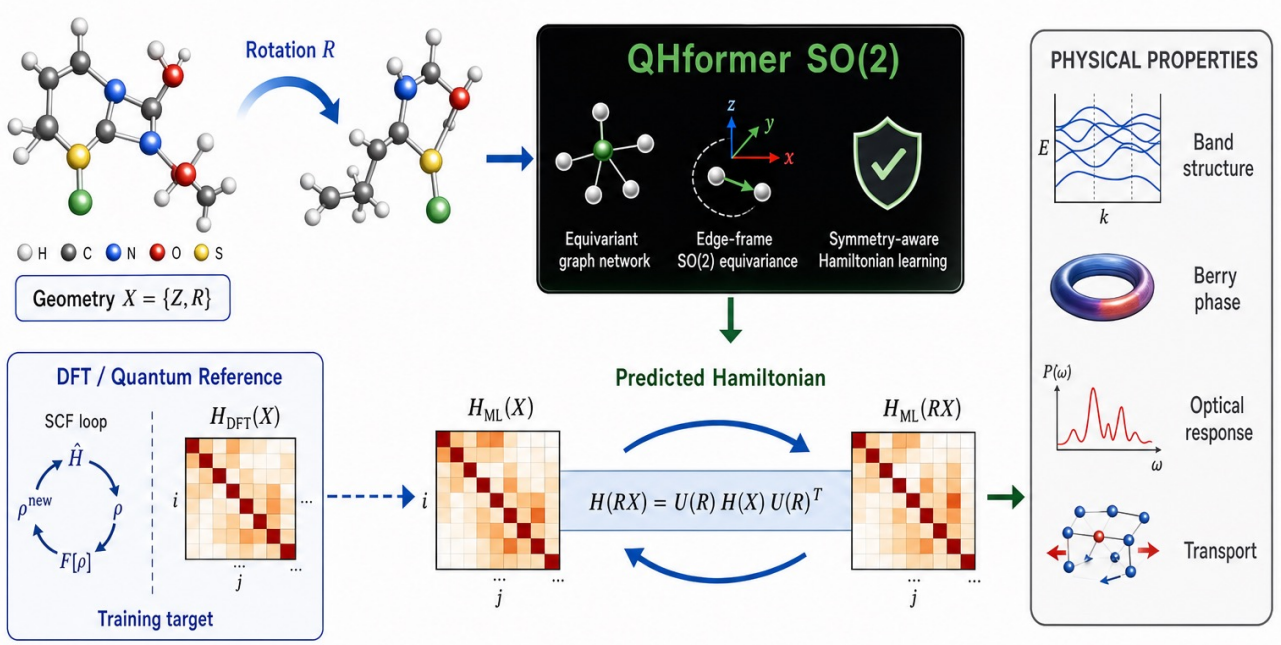


- **Full Differentiable TDDFT Code for Neural Network XC Functional**
- **Qhformer_V2-A Sparsity Attention Hamiltonian Learning Model**
- **DL-TDDFT – A Code for Accelerating TDDFT**
- **Quantum Chemistry with OpenClaw, Hermes – Automatic Materials Calculation Skills and Agents Team for Researching**

GradTDDFT

AI-native differentiable TDDFT in JAX





■ Background – Neural Network Training and Automatic Differentiable Program

● Basic Theory of Differentiation

- **Analytical Differentiation**

  Derived purely by hand based on the derivative formula, then coding

- **Numerical Differentiation**

  $$f'(x_i) \approx \frac{f(x_i) - f(x_i - \Delta)}{\Delta}$$

  Difference approximates derivatives

- **Symbolic Differentiation**

  Manual differentiation coding is implemented based on certain recursive relationships

- **Automatic Differentiation**

  Based on the chain rule for derivative of composite functions, operators propagate gradients step by step on the computation graph

● Basic Theory of Automatic Differentiation

Think of the function $y = (f_k \circ f_{k-1} \circ f_{k-2} \circ \dots \circ f_1 \circ f_0)(x)$ and the derivation on $x = x_0$ of y

- **Forward Mode**

  $x_0 = x$ and $x_1 = f_0(x_0)\ x_2 = f_1(x_1) \dots x_{i+1} = f_i(x_i)$

  $x_0' = 1$

  $x_{i+1}' = f_i'(x_i)x_i'$

  $y' = x_{i+1}'$

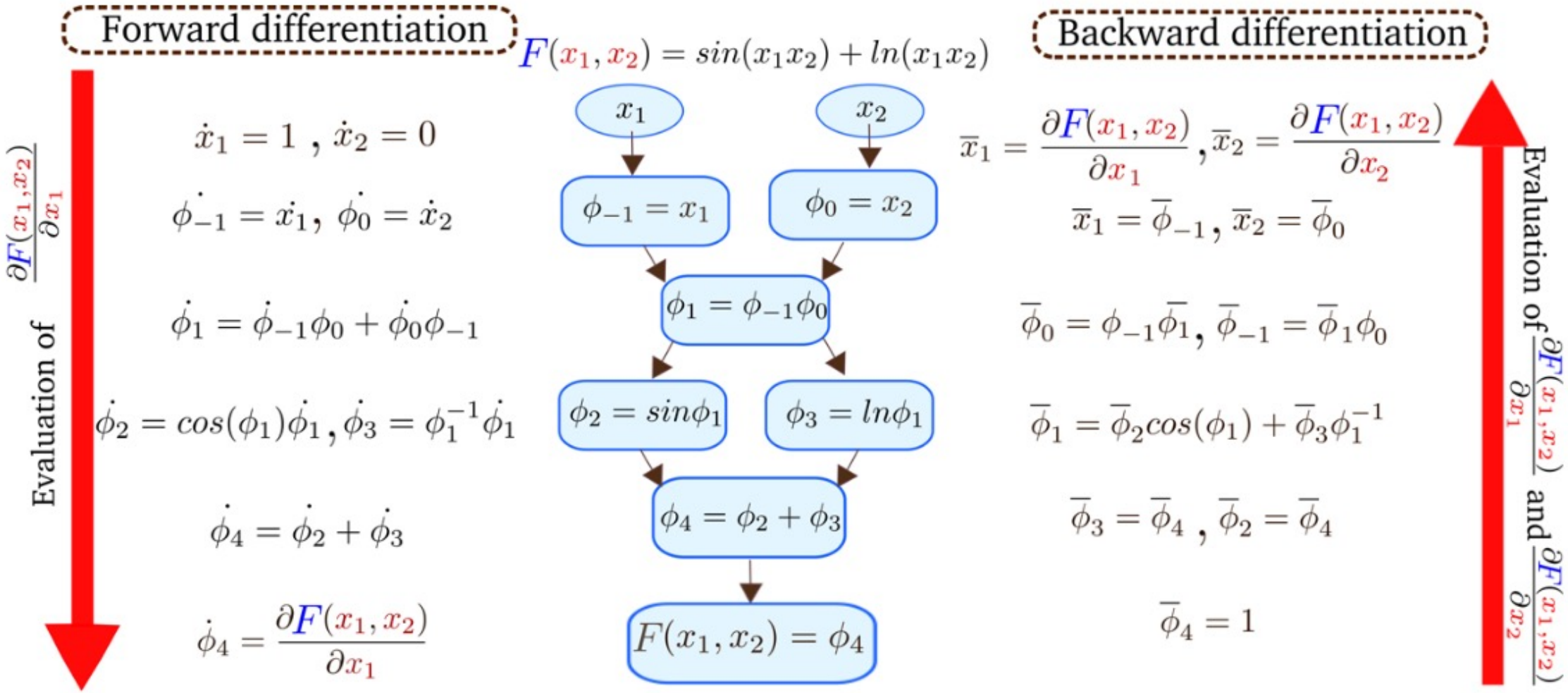
- **Backward Mode**

  $\bar{x}_{k+1} = 1$

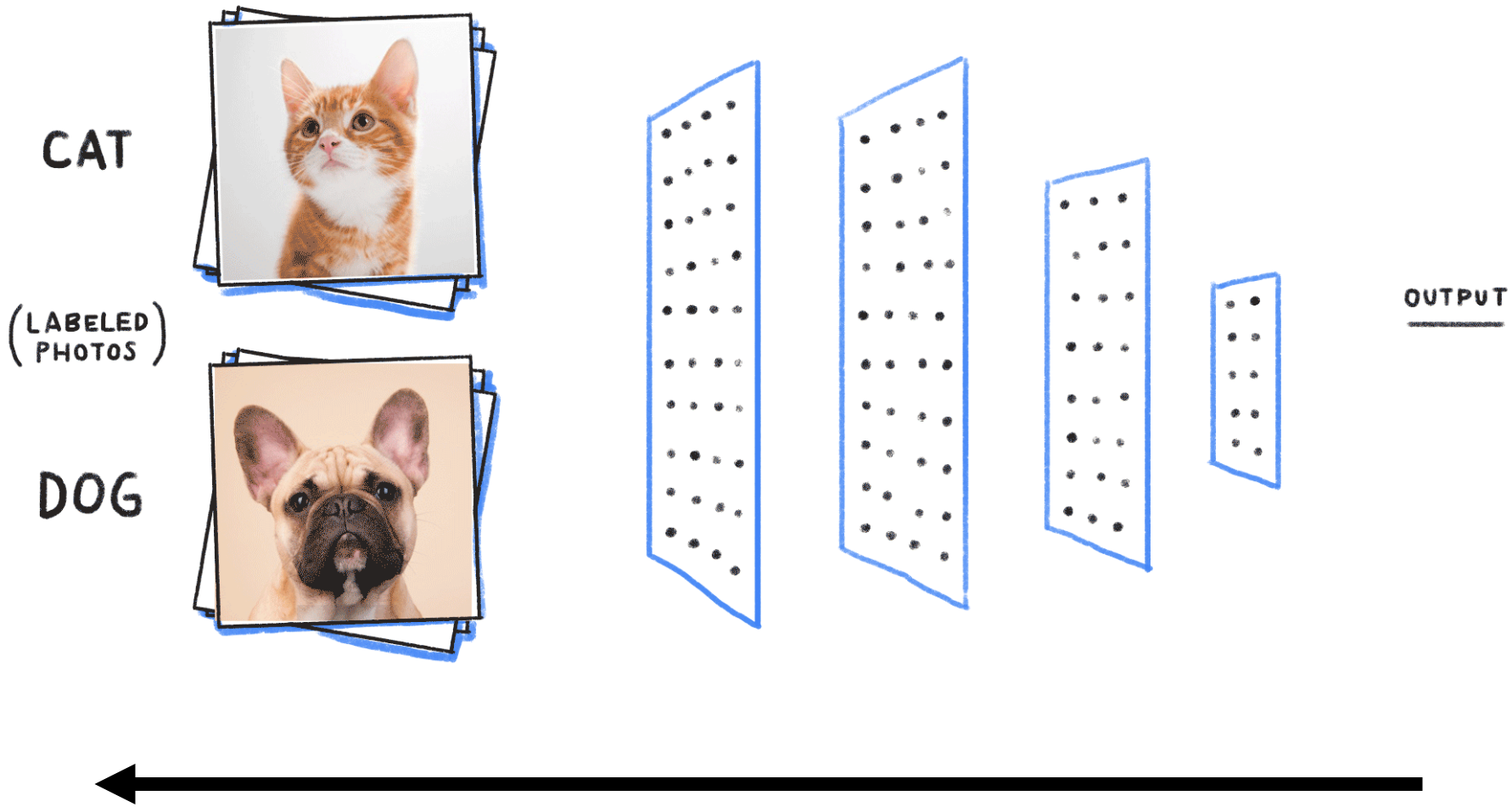
  $\bar{x}_i = \bar{x}_{i+1} \cdot f_i'(x_i)\ with\ i = k, k-1, \dots, 1,0$

  $\bar{x}_0 = y'$

■ Background – Neural Network Training and Automatic Differentiable Program



■ Background – Neural Network Training and Automatic Differentiable Program



■ Background – Neural Network Training and Automatic Differentiable Program

- **Neural Networ Forward Mode**

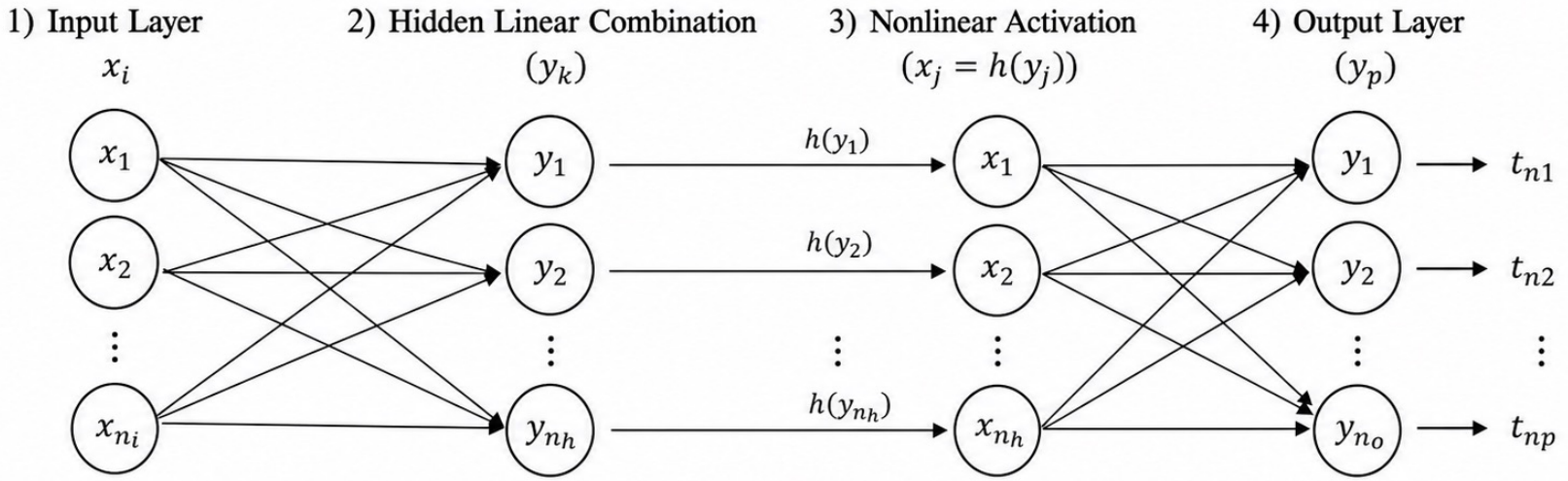
Single-Layer Network

$$y_k = \sum_i \omega_{ki} x_i \qquad x_j = h(y_k)$$

$$y_p = \sum_j \omega_{pj} x_j$$

With Loss Function

$$L_n = \frac{1}{2} \sum_p (y_{np} - t_{nk})^2$$

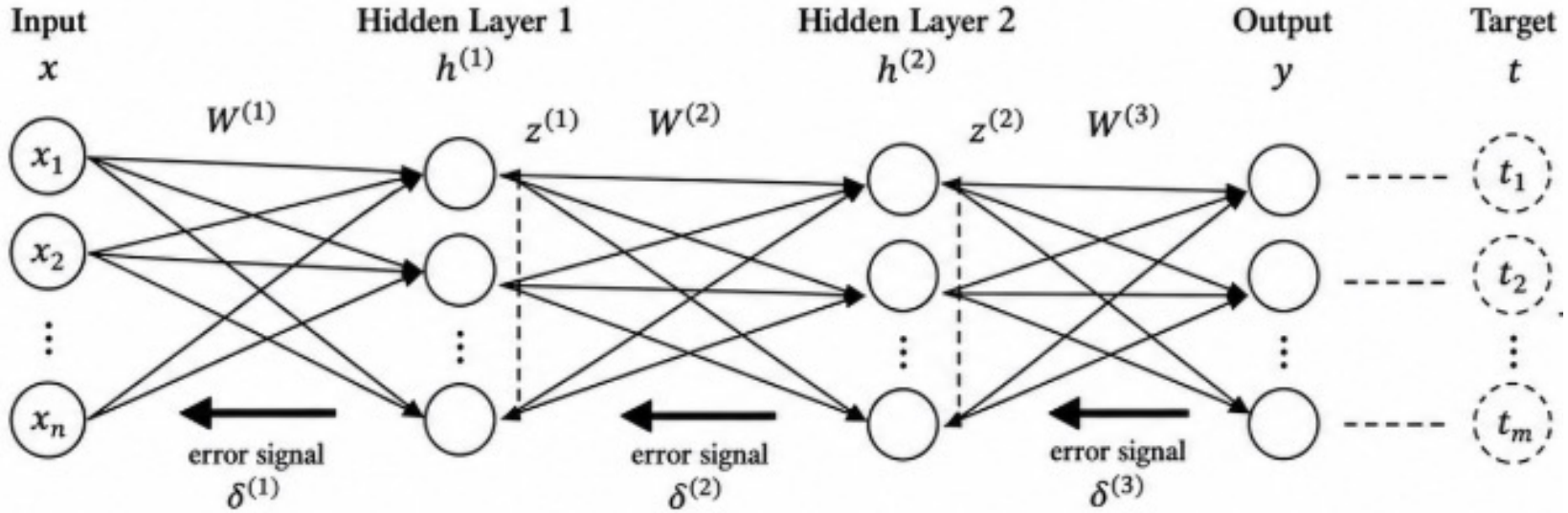


- **Neural Networ Backward Mode**

$$\frac{\partial L_n}{\partial \omega_{ki}} = \frac{\partial L_n}{\partial y_k} \frac{\partial y_k}{\partial \omega_{ki}} = \delta_k x_i$$

$$\delta_k = \frac{\partial L_n}{\partial y_k} = \sum_p \frac{\partial L_n}{\partial y_p} \frac{\partial y_p}{\partial y_k}$$

$$\sum_p \frac{\partial L_n}{\partial y_p} \frac{\partial y_p}{\partial x_j} \frac{\partial x_j}{\partial y_k} = h'(y_k) \sum_p \delta_p \omega_{pj}$$



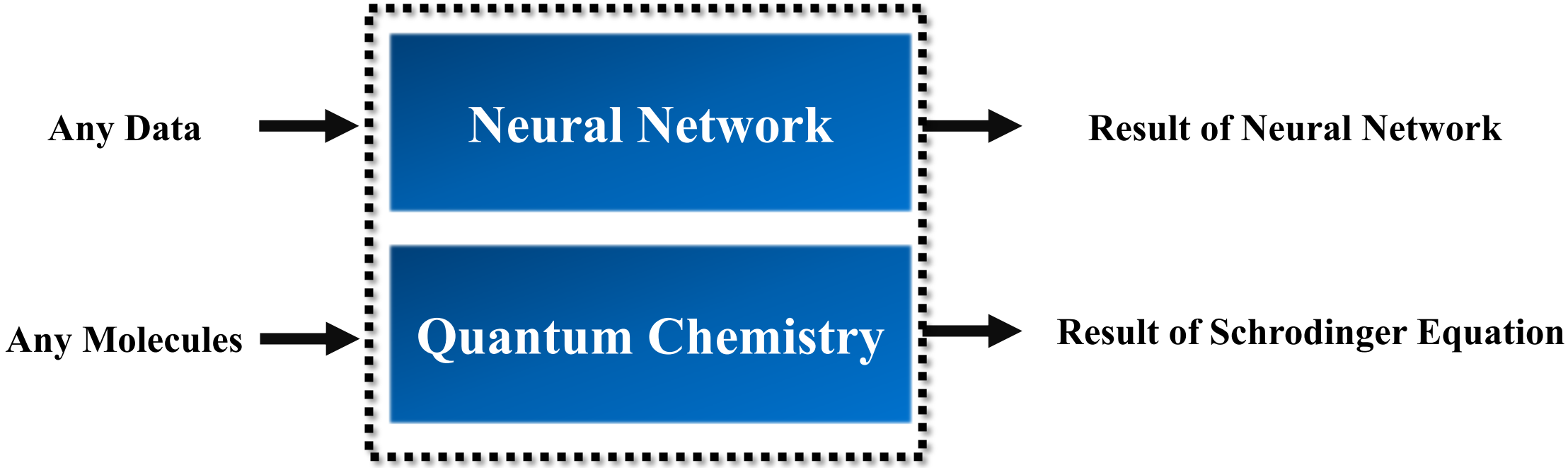
■ Background – Neural Network Training and Automatic Differentiable Program

自动微分与量子化学--搞一个可微分的SCF程序

Yuan

中国科学院大学 物理学博士在读

Post-Ken 等 40 人赞同了该文章 ›



zhuanlan.zhihu.com/p/203755074962273590 1

■ Background – SIE and TDDFT

$$v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\}) \Leftrightarrow \rho(\boldsymbol{x}) \Leftrightarrow \{\psi_i(\boldsymbol{x})\}$$

$$\left[-\frac{1}{2}\nabla^2 + v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\}) + v_H(\boldsymbol{x}) + v_{xc}(\boldsymbol{x})\right]\psi_i(\boldsymbol{x}) = \varepsilon_i\psi_i(\boldsymbol{x})$$

Runge-Gross Theorem

$$v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\},\boldsymbol{t}) + \boldsymbol{C}(\boldsymbol{t}) \Leftrightarrow \rho(\boldsymbol{x},\boldsymbol{t}) \Leftrightarrow \{\psi_i(\boldsymbol{x},\boldsymbol{t})\boldsymbol{e}^{-\boldsymbol{i}\boldsymbol{\alpha}(\boldsymbol{t})}\}$$

$$\left[-\frac{1}{2}\nabla^2 + v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\},\boldsymbol{t}) + v_H(\boldsymbol{x},\boldsymbol{t}) + v_{xc}(\boldsymbol{x},\boldsymbol{t})\right]\psi_i(\boldsymbol{x},\boldsymbol{t}) = i\frac{\partial}{\partial t}\psi_i(\boldsymbol{x},\boldsymbol{t})$$

The Time-Dependent of XC Functional Vanished !

Linear-Response – Adiabatic Approximation

$$F_{aa}^{(0)}x_{ai} - x_{ai}F_{ii}^{(0)} + \sum_{bj}\left(\frac{\partial F_{ai}}{\partial P_{bj}}x_{bj} + \frac{\partial F_{ai}}{\partial P_{jb}}y_{bj}\right)P_{ii}^{(0)} = \omega x_{ia}$$

$$F_{aa}^{(0)}y_{ai} - y_{ai}F_{ii}^{(0)} - \sum_{bj}P_{ii}^{(0)}\left(\frac{\partial F_{ia}}{\partial P_{bj}}x_{bj} + \frac{\partial F_{ia}}{\partial P_{jb}}y_{bj}\right) = \omega x_{ia}$$

$$F_{aa}^{(0)} = \int d\boldsymbol{r}\psi_a^*(\boldsymbol{x},\boldsymbol{t})\left\{-\frac{1}{2}\nabla^2 + v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\},\boldsymbol{t}) + v_H(\boldsymbol{x},\boldsymbol{t}) + v_{xc}(\boldsymbol{x},\boldsymbol{t})\right\}\psi_a(\boldsymbol{x},\boldsymbol{t})$$

$$F_{ia} = \int d\boldsymbol{r}\psi_i^*(\boldsymbol{x},\boldsymbol{t})\left\{-\frac{1}{2}\nabla^2 + v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\},\boldsymbol{t}) + v_H(\boldsymbol{x},\boldsymbol{t}) + v_{xc}(\boldsymbol{x},\boldsymbol{t})\right\}\psi_a(\boldsymbol{x},\boldsymbol{t}) + g_{ai}(\omega) + \Delta F_{ia}^{(0)}$$

Casida-TDDFT

$$\begin{pmatrix} A & B \\ B^* & A^* \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix} = \omega\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix}$$

$$\psi_{ex} = \sum_{ai} x_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$

$$\psi_{di} = \sum_{ai} y_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$

TDA-TDDFT

$$AX = \omega X$$

$$\psi_{ex} = \sum_{ai} x_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$

■ Background – SIE and TDDFT

A, B Matrix and Response Kernel

$$A = \delta_{ij}\delta_{ab}(\varepsilon_a - \varepsilon_i) + \int d\boldsymbol{r}\int \boldsymbol{r}'\phi_i(\boldsymbol{r})\phi_a^*(r)\left\{\frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|} + f_{xc}\right\}\phi_b^*(r')\phi_j(\boldsymbol{r}')$$

$$B = \int d\boldsymbol{r}\int \boldsymbol{r}'\phi_i(\boldsymbol{r})\phi_a^*(r)\left\{\frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|} + f_{xc}\right\}\phi_b^*(r')\phi_j(\boldsymbol{r}')$$

$$f_{xc}[\rho] = \frac{\delta^2 E_{xc}[\rho]}{\delta\rho(\boldsymbol{r})\delta\rho(\boldsymbol{r}')}$$

TDA-TDDFT

$$AX = \omega X$$

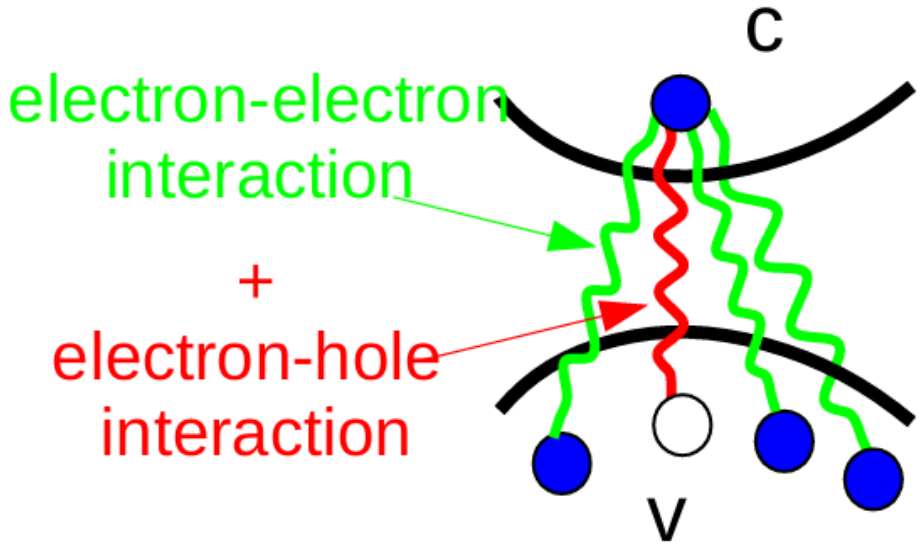
$$\psi_{ex} = \sum_{ai} x_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$

Casida-TDDFT

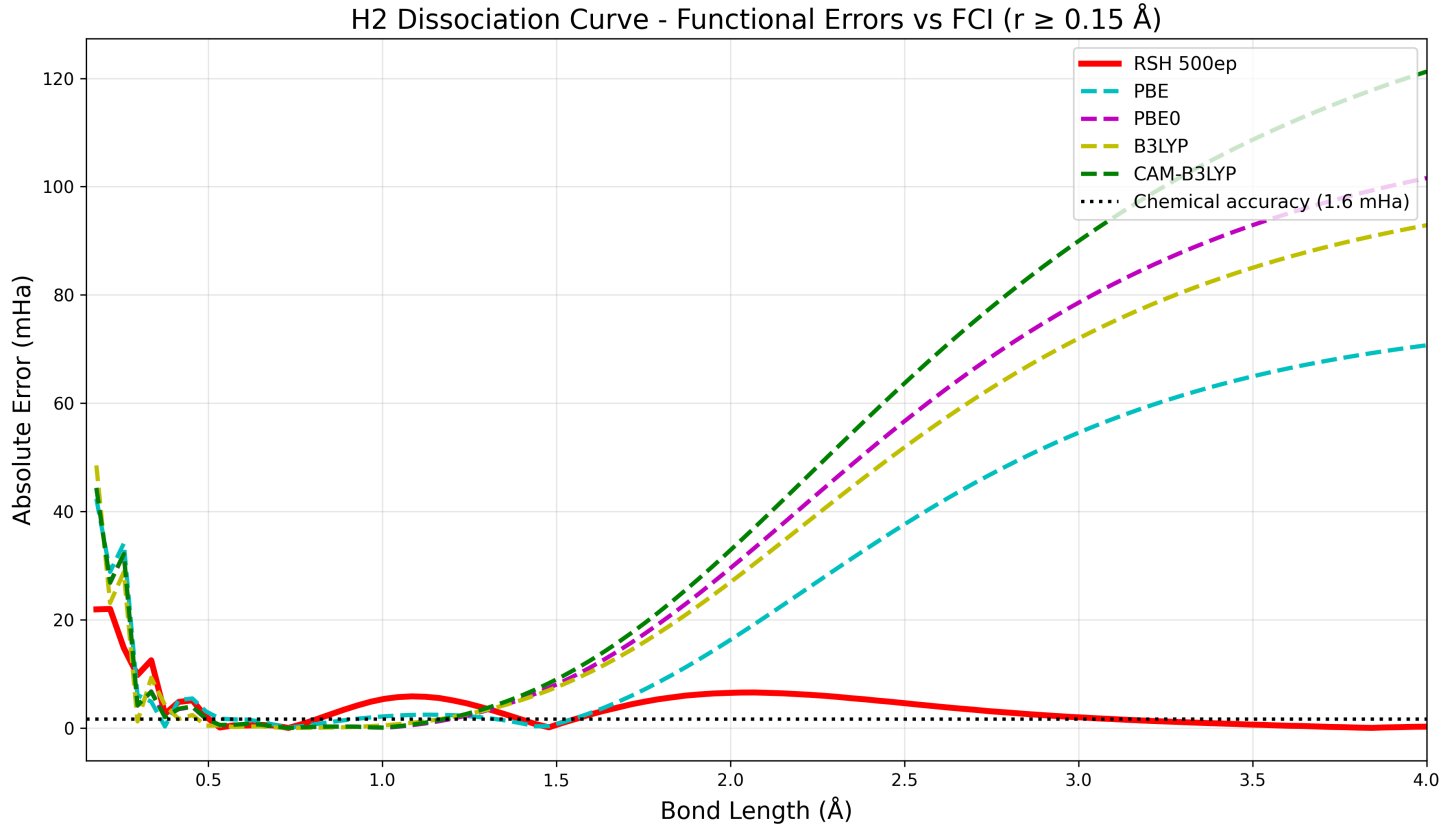
$$\begin{pmatrix} A & B \\ B^* & A^* \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix} = \omega\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix}$$

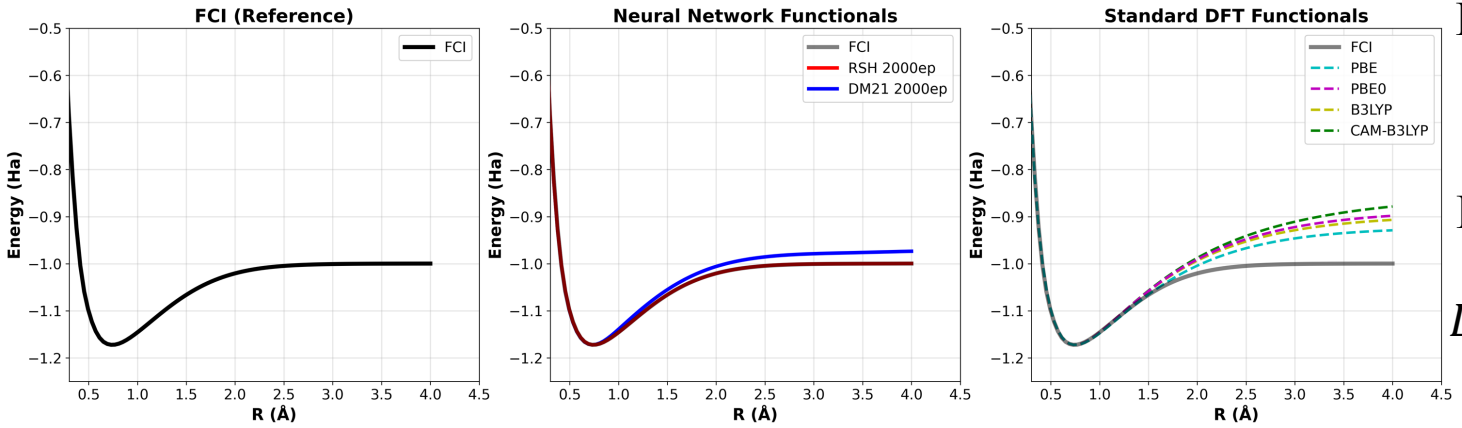
$$\psi_{ex} = \sum_{ai} x_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$

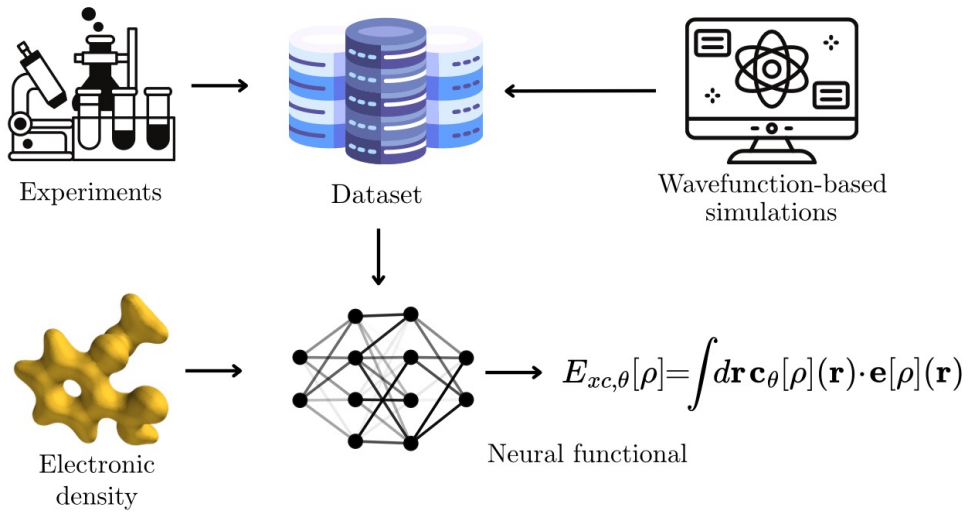
$$\psi_{di} = \sum_{ai} y_{ai}\hat{a}_a^+\hat{a}_i\psi_{gr}$$



■ GradTDDFT – GradDFT for Ground State







J. Chem. Phys. 160, 062501 (2024)

Coeffcient Functions

$$c_\theta[\rho](\boldsymbol{r}) = (c_{1,\theta}[\rho](\boldsymbol{r}), c_{2,\theta}[\rho](\boldsymbol{r}) \dots c_{k,\theta}[\rho](\boldsymbol{r}))$$

Energy Densities

$$e[\rho](\boldsymbol{r}) = (e_1[\rho](\boldsymbol{r}), e_2[\rho](\boldsymbol{r}) \dots e_k[\rho](\boldsymbol{r}))$$

Loss Function

$$L = \frac{1}{N^2}\sum_i \left(E_{pre} - E_{true}\right)^2 + \frac{1}{N^2}\sum_i \left(\rho_{pre} - \rho_{true}\right)^2$$

■ GradTDDFT – Response Kernel with Automatic Differentiation

- **Total XC Energy**

$$E_{xc}^{\theta}[\rho(\boldsymbol{r})] = \int d\boldsymbol{r} e_{xc}^{\theta}[\rho(\boldsymbol{r})]$$

- **XC Energy on grid** $e_{xc}^{\theta}[\rho(\boldsymbol{r})]$

- **XC Potential on Grid**

$$v_{xc}^{\theta}[\rho(\boldsymbol{r})] = \frac{\partial e_{xc}^{\theta}[\rho(\boldsymbol{r})]}{\partial \boldsymbol{\rho}(r)}$$

$$f_{xc}^{\theta}[\rho(\boldsymbol{r})] = \frac{\partial^2 e_{xc}^{\theta}[\rho(\boldsymbol{r})]}{\partial \boldsymbol{\rho}^2(r)}$$

- **Calculated by jax**

jax.grad()/jax.hessian()

$$e_{xc}^{\theta}[\rho(\boldsymbol{r})] \longrightarrow v_{xc}^{\theta}[\rho(\boldsymbol{r})] \longrightarrow f_{xc}^{\theta}[\rho(\boldsymbol{r})]$$

```python
def point_energy(variables: Array) -> Array:
    rho_point = jnp.maximum(variables[0],
density_floor_value)
    if kind_norm == "LDA":
        grad_point = jnp.zeros((3,),
dtype=variables.dtype)
        tau_point = jnp.asarray(0.0,
dtype=variables.dtype)
    elif kind_norm == "GGA":
        grad_point = variables[1:4]
        tau_point = jnp.asarray(0.0,
dtype=variables.dtype)
    elif kind_norm == "MGGA":
        grad_point = variables[1:4]
        tau_point = jnp.maximum(variables[4], 0.0)
    ...
    return eval_xc_energy_density(spec_norm, features)

return jax.jit(jax.vmap(jax.hessian(point_energy)))
```

■ GradTDDFT – Response Kernal with Automatic Differentiation

- **LDA**

$$E_{xc}[\rho(\boldsymbol{r})] = \int d\boldsymbol{r} e_{xc}[\rho(\boldsymbol{r})]$$

$$f_{xc}[\rho(\boldsymbol{r})] = \frac{\partial^2 e_{xc}}{\partial \rho^2}$$

- **GGA**

$$E_{xc} = \int d\boldsymbol{r} e_{xc}[\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r})]$$

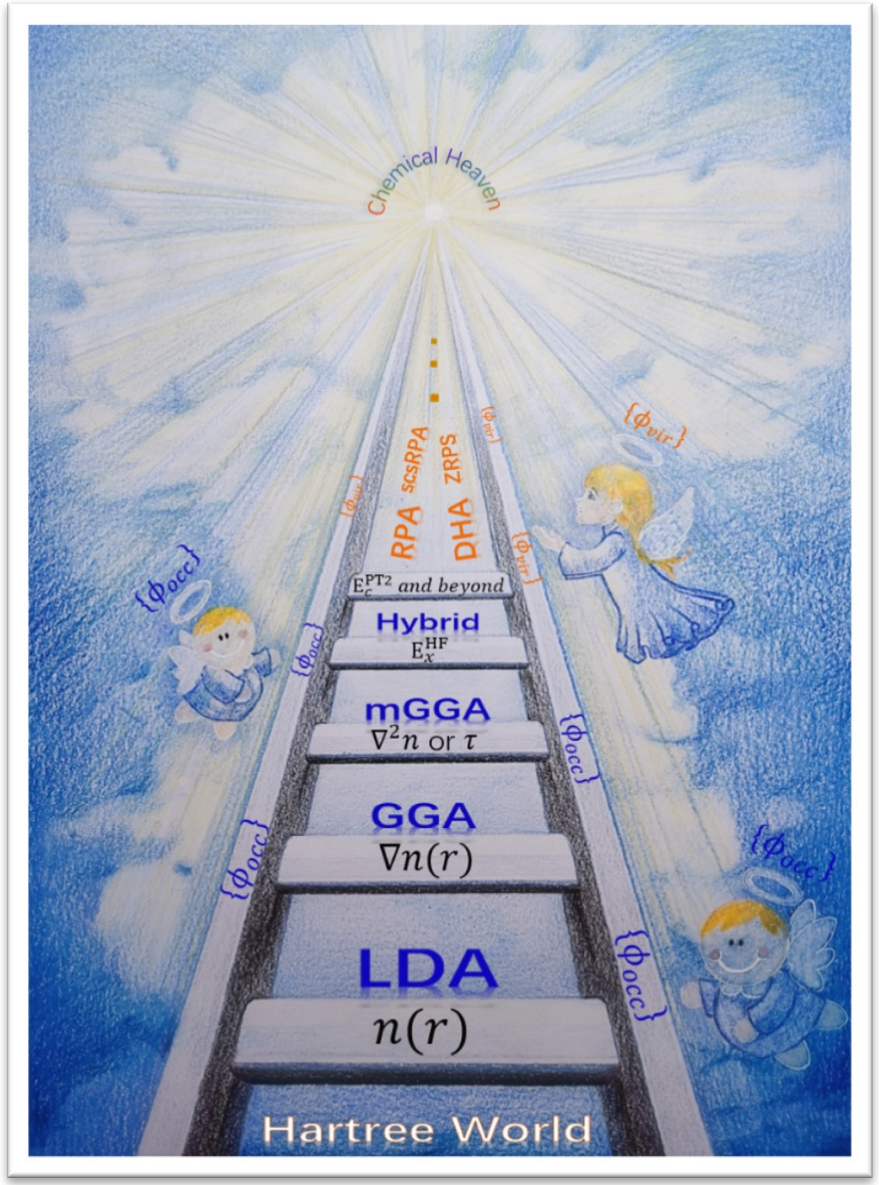
$$\delta E_{xc} = \int d\boldsymbol{r} \delta e_{xc}[\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r})] = \int d\boldsymbol{r} \frac{\partial e_{xc}}{\partial \rho}\delta\rho + \frac{\partial \nabla e_{xc}}{\partial \nabla\rho}\delta\nabla\rho$$

$$\delta^2 E_{xc} = \int d\boldsymbol{r} \sum_{\mu\nu} \frac{\partial^2 e_{xc}}{\partial u_\nu \partial u_\mu}\delta u_\nu \delta u_\mu$$

- **Meta-GGA**

$$E_{xc} = \int d\boldsymbol{r} e_{xc}[\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r}), \tau] \qquad \tau = -\frac{1}{2}\sum_i |\nabla\psi|^2$$

$$f_{xc} = \frac{\partial^2 e_{xc}}{\partial u_\nu \partial u_\mu} \qquad u_\mu = [\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r}), \tau]$$



■ GradTDDFT– Response Kernel with Automatic Differentiation

- **HF Exchange Kernel**

$$E_x^{HF} = \int d\boldsymbol{r} c_x^{hf}[\rho(\boldsymbol{r})] e_x^{hf}[\phi(\boldsymbol{r})]$$

$$e_x^{hf}[\phi] = -\frac{1}{2}\sum_{pq} \phi_p^*(r)\phi_q(r) \int d\boldsymbol{r}' \frac{\phi_i^*(\boldsymbol{r}')\phi_j(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}$$

In A matrix of Casida equation

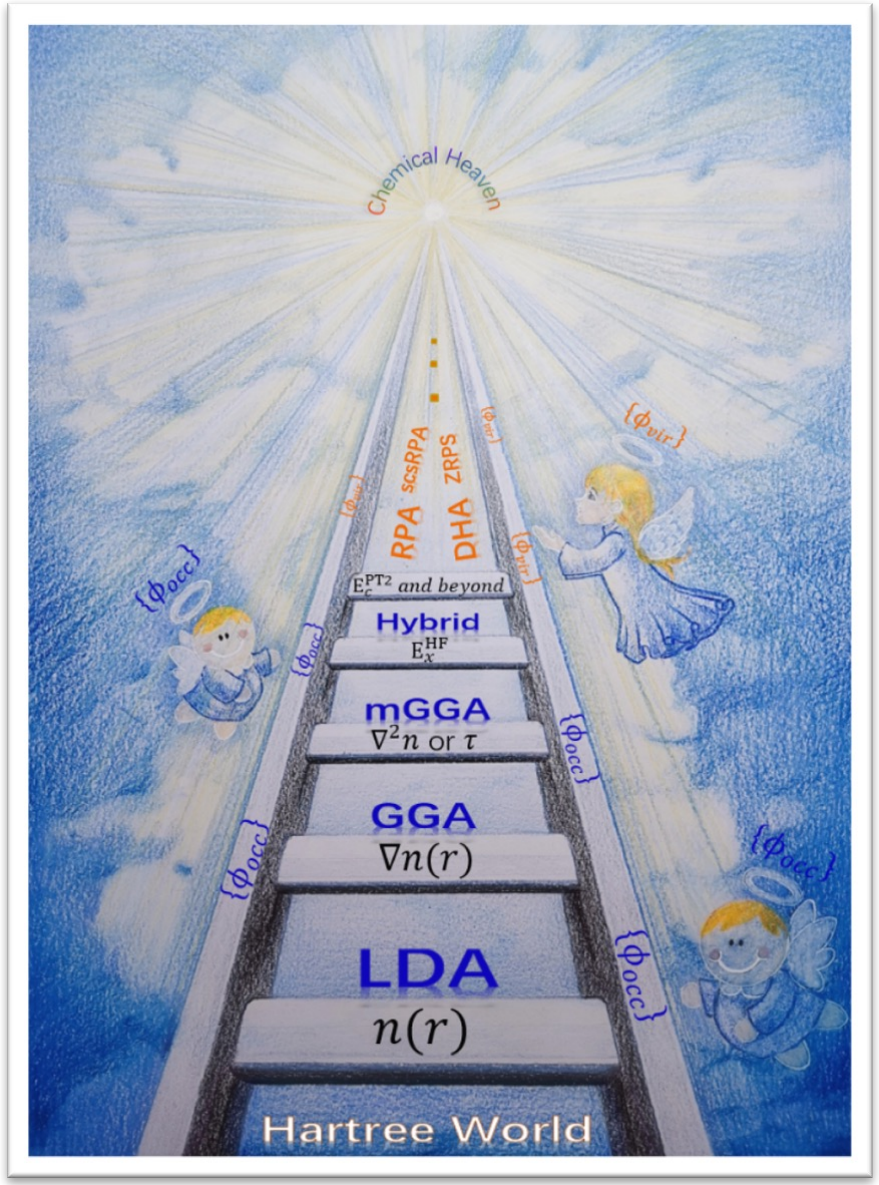
$$K_{ai,bj} = -\alpha_{eff}(ij|ab) \quad \text{with} \quad \alpha_{eff} = \frac{\int c_x^{hf}[\rho(\boldsymbol{r})]\rho(\boldsymbol{r})d\boldsymbol{r}}{\int \rho(\boldsymbol{r})d\boldsymbol{r}}$$

- **PT2 kernel**

$$E_c^{PT2} = \sum_{ia,jb} \frac{(ai|jb)[2(ai|jb)-(ib|ja)]}{\epsilon_i+\epsilon_j-\epsilon_a-\epsilon_b} \rightarrow \int d\boldsymbol{r} c_c^{pt2}[\rho] e_c^{pt2}[\phi(\boldsymbol{r})]$$

$$e_c^{PT2} = \sum_{ia,jb} \rho_{ai} V_{jb}(\boldsymbol{r}) \frac{[2(ai|jb)-(ib|ja)]}{\epsilon_i+\epsilon_j-\epsilon_a-\epsilon_b}$$

$$f_{xc}^{pt2} = \frac{\partial^2 c_c^{pt2}[\rho]}{\partial u_\nu \partial u_\mu} e_c^{pt2}[\phi(\boldsymbol{r})] + \frac{\partial^2 e_c^{pt2}[\phi(\boldsymbol{r})]}{\partial u_\nu \partial u_\mu} c_c^{pt2}[\rho]$$



■ GradTDDFT– Response Kernel with Automatic Differentiation

- **Approxmated PT2 kernel For Double Hybrid XC Functional In TDDFT**

$$E_c^{PT2} = \sum_{ia,jb} \frac{(ai|jb)[2(ai|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \rightarrow \int d\boldsymbol{r} c_c^{pt2}[\rho] e_c^{pt2}[\phi(\boldsymbol{r})]$$

$$e_c^{PT2} = \sum_{ia,jb} \rho_{ai} V_{jb}(\boldsymbol{r}) \frac{[2(ai|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

$$f_{xc}^{pt2} = \frac{\partial^2 c_c^{pt2}[\rho]}{\partial u_\nu \partial u_\mu} e_c^{pt2}[\phi(\boldsymbol{r})] + \frac{\partial^2 e_c^{pt2}[\phi(\boldsymbol{r})]}{\partial u_\nu \partial u_\mu} c_c^{pt2}[\rho] = \frac{\partial^2 c_c^{pt2}[\rho]}{\partial u_\nu \partial u_\mu} e_c^{pt2}[\phi(\boldsymbol{r})] \quad \rightarrow \quad \textbf{Fixed Local PT2 Feature}$$

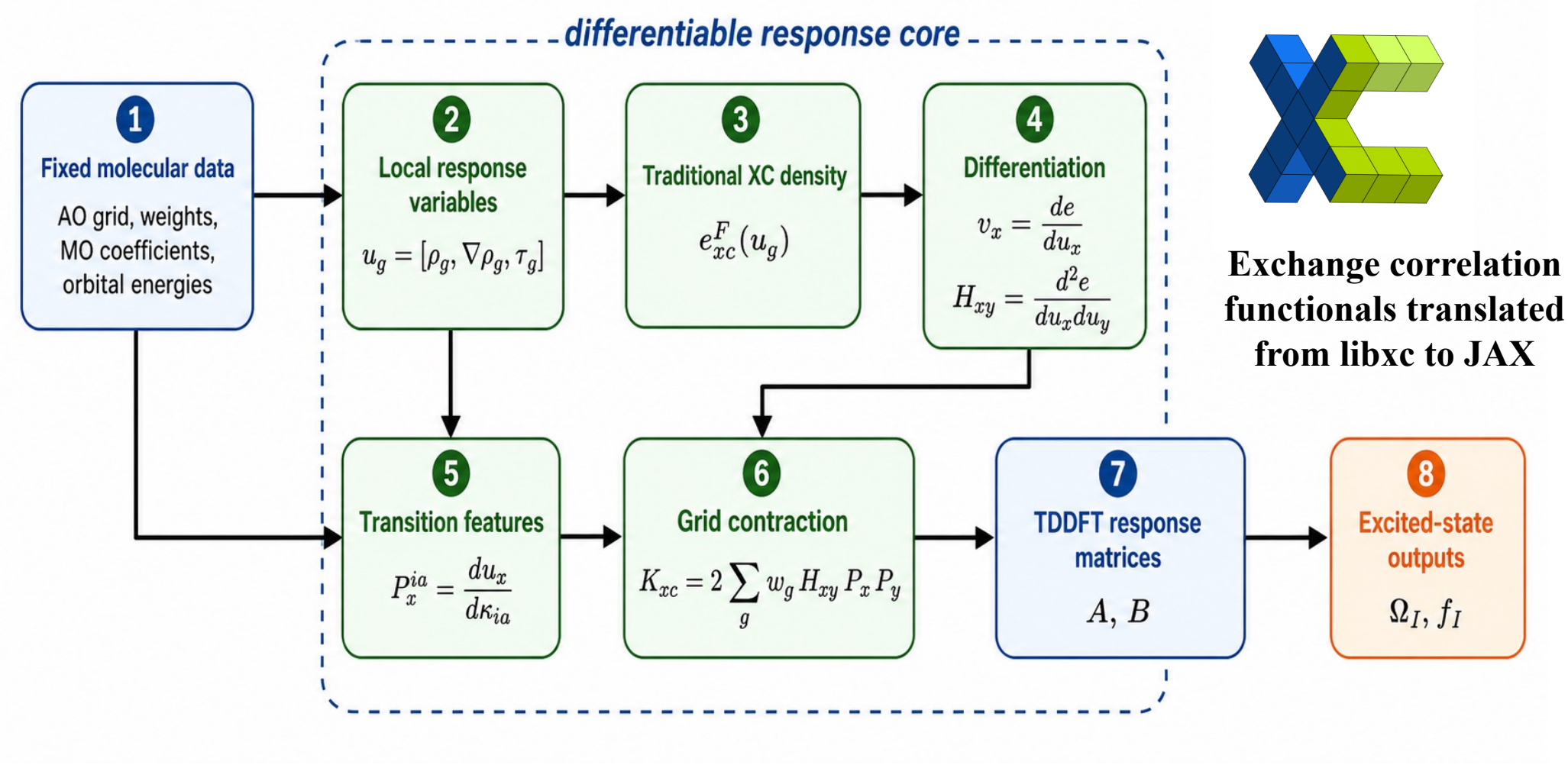
- **CIS(D) For Double Hybrid XC Functional In TDDFT**

$$\begin{pmatrix} A & B \\ B^* & A^* \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \omega \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} \qquad \textbf{With } B = X + Y$$

$$\omega_{PT2} = \omega + \mathrm{a_c^\theta} \Delta^{PT2} \qquad \mathrm{a_c^\theta} = \frac{\int c_c^{pt2}[\rho] \rho(\boldsymbol{r}) d\boldsymbol{r}}{\int \rho(\boldsymbol{r}) d\boldsymbol{r}}$$

$$\Delta^{PT2} = -\frac{1}{4} \sum_{ijab} \frac{U_{ijab}^I}{\varepsilon_a + \varepsilon_b - \varepsilon_j - \varepsilon_i - \omega} \qquad U_{ijab}^I = \sum_c (ab|cj) B_{ic} - (ab|ci) B_{jc} + \sum_k (ka|ij)\, B_{kb} - (kb|ij) B_{ka}$$

■ GradTDDFT – Differentiable TDDFT Kernel



Exchange correlation functionals translated from libxc to JAX

Github.com/STOKES-DOT/GradTDDFT

■ GradTDDFT – Differentiable TDDFT Kernel

Excitation-energy and oscillator-strength agreement

Molecule	TDA MAE(Ω) / eV	TDA MAE(f)	TDDFT MAE(Ω) / eV	TDDFT MAE(f)
H2O	1.31e-11	8.37e-09	2.97e-11	1.76e-08
CO	3.62e-13	7.22e-09	6.84e-12	2.82e-08
N2	5.86e-14	1.86e-22	3.72e-13	3.26e-22
C2H4	1.09e-11	3.13e-09	1.72e-13	9.22e-10
CH2O	1.97e-12	8.04e-10	8.25e-13	3.11e-10
C6H6	2.87e-10	5.89e-12	6.98e-11	1.29e-07

Excitation-energy and oscillator-strength agreement

Molecule	TDA MAE(Ω) / eV	TDA MAE(f)	TDDFT MAE(Ω) / eV	TDDFT MAE(f)
H2O	1.31e-11	8.37e-09	2.97e-11	1.76e-08
CO	3.62e-13	7.22e-09	6.84e-12	2.82e-08
N2	5.86e-14	1.86e-22	3.72e-13	3.26e-22
C2H4	1.09e-11	3.13e-09	1.72e-13	9.22e-10
CH2O	1.97e-12	8.04e-10	8.25e-13	3.11e-10
C6H6	2.87e-10	5.89e-12	6.98e-11	1.29e-07

PBE/def2-SVP, restricted singlet, first five states per molecule.

■ GradTDDFT – Training Modes in Neural Network XC Functional

- **Ground State**

Coeffcient Functions

$$c_\theta[\rho](\boldsymbol{r}) = (c_{1,\theta}[\rho](\boldsymbol{r}), c_{2,\theta}[\rho](\boldsymbol{r}) \dots c_{k,\theta}[\rho](\boldsymbol{r}))$$

Energy Densities

$$e[\rho](\boldsymbol{r}) = (e_1[\rho](\boldsymbol{r}), e_2[\rho](\boldsymbol{r}) \dots e_k[\rho](\boldsymbol{r}))$$

Loss Function

$$L = \frac{1}{N^2}\sum_i \left(E_{pre} - E_{true}\right)^2 + \frac{1}{N^2}\sum_i \left(\rho_{pre} - \rho_{true}\right)^2$$

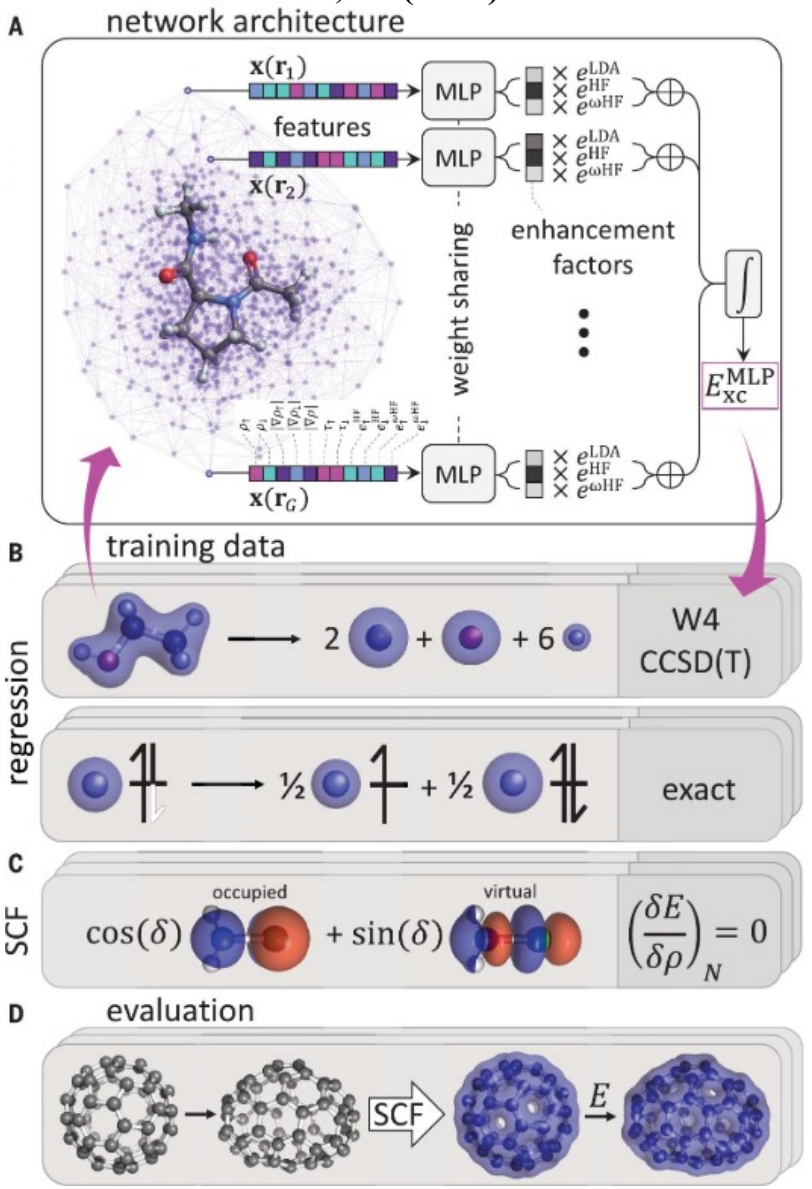
- **Excited State**

jax.grad(XC)/jax.hessian(XC) on Grids

$$c_\theta[\rho](\boldsymbol{r}) e^\theta_{xc}[\rho(\boldsymbol{r})] \longrightarrow v^\theta_{xc}[\rho(\boldsymbol{r})] \longrightarrow f^\theta_{xc}[\rho(\boldsymbol{r})]$$

$$L = \frac{1}{N^2}\sum_i \left(E_{pre} - E_{true}\right)^2 + \frac{1}{N}\sum_i \left(E_{pre} - E_{true}\right)$$

Science. 2021;374(6573):1385-1389



■ GradTDDFT – Training Modes in Neural Network XC Functional

- **Fixed-Density Mode**

$$D_\theta = D_{\text{ref}}$$

$$L = L(\theta, D)\Big|_{D=D_{ref}}$$

$$\nabla_\theta L = \partial_\theta L(\theta, D)\Big|_{D=D_{ref}} \quad \text{with } \partial_\theta D = 0$$

- **Explicit SCF Differential Mode**

$$L = L(\theta, D_\theta) \quad \text{with } D_\theta = T^\theta_{SCF}(D_0)$$

$$\nabla_\theta L = \partial_\theta L(\theta, D_\theta) + \partial_{D_\theta} L(\theta, D_\theta) \cdot \partial_\theta D_\theta$$

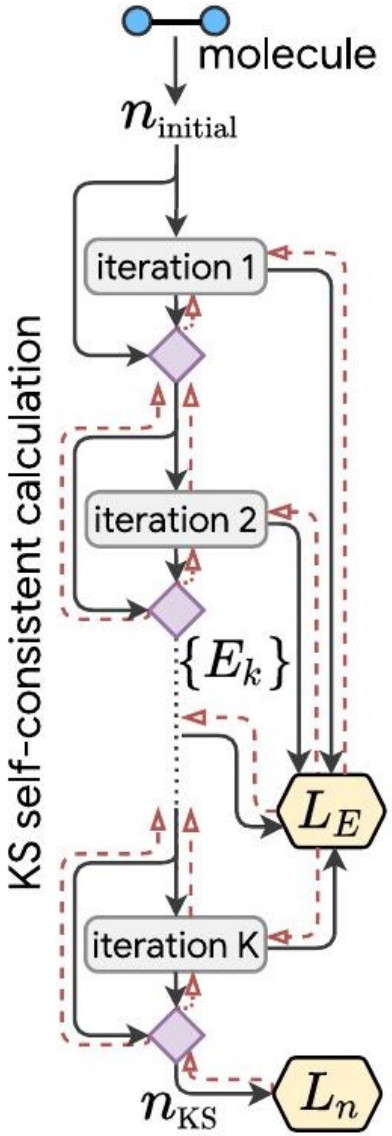
$$\text{with } \partial_\theta D_\theta = \textstyle\sum_{K-1}[\Pi^{K-1}_{s=t+1} \partial_D T^\theta_{SCF}(D^s)]\partial_\theta T^\theta_{SCF}(D^t)$$

- **Implicit SCF Differential Mode**

$$L = L(\theta, D_\theta) \quad \text{with } D_\theta = T^K_{SCF}(D_0)$$

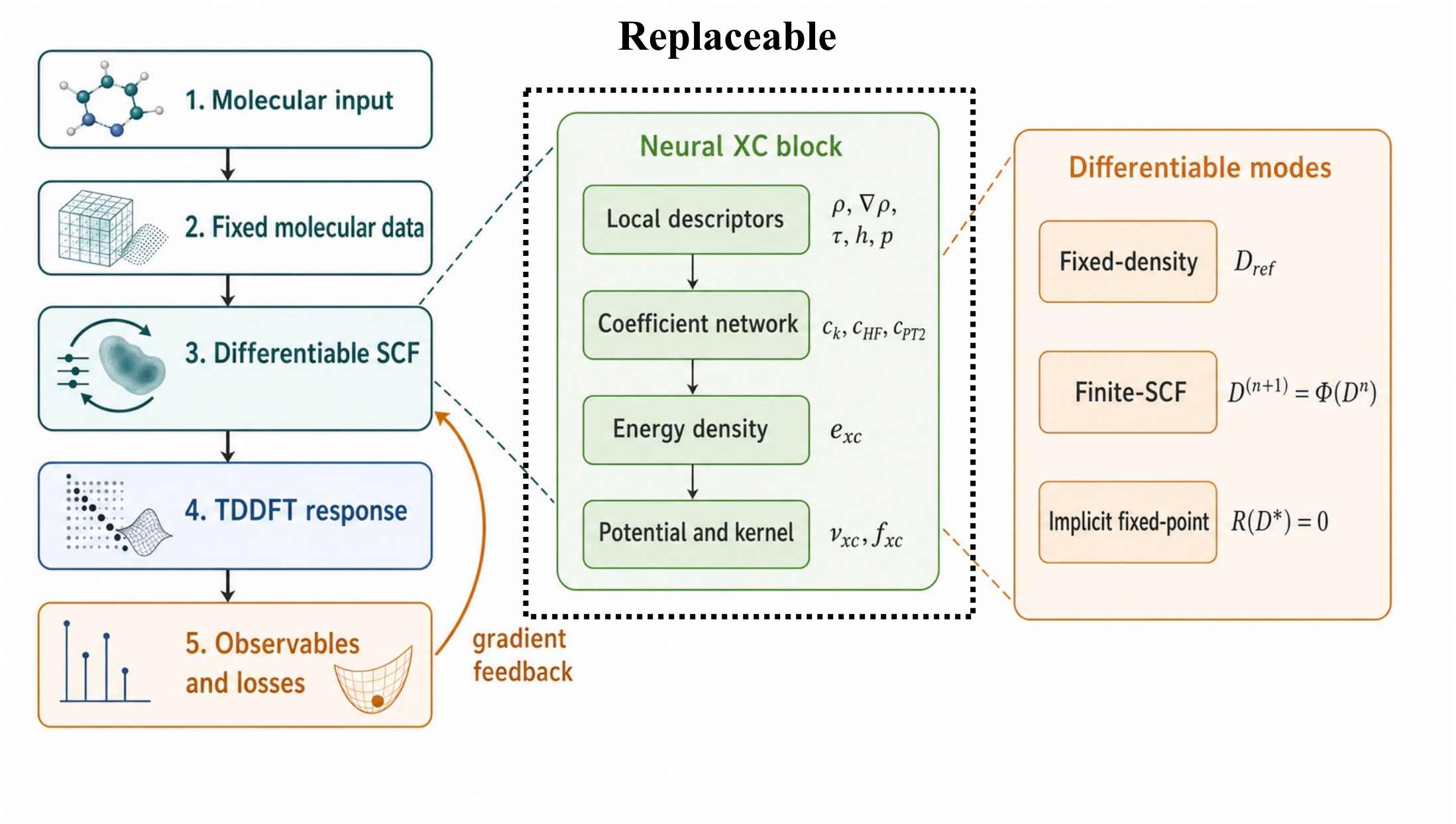
$$R(D_\theta, \theta) = F_\theta(D_\theta)D_\theta S - SD_\theta F_\theta(D_\theta) = 0$$

$$\nabla_\theta L = \partial_\theta L(\theta, D_\theta) + \partial_{\rho_\theta} L(\theta, D_\theta)\frac{\partial_\theta R}{\partial_{D_\theta} R}$$

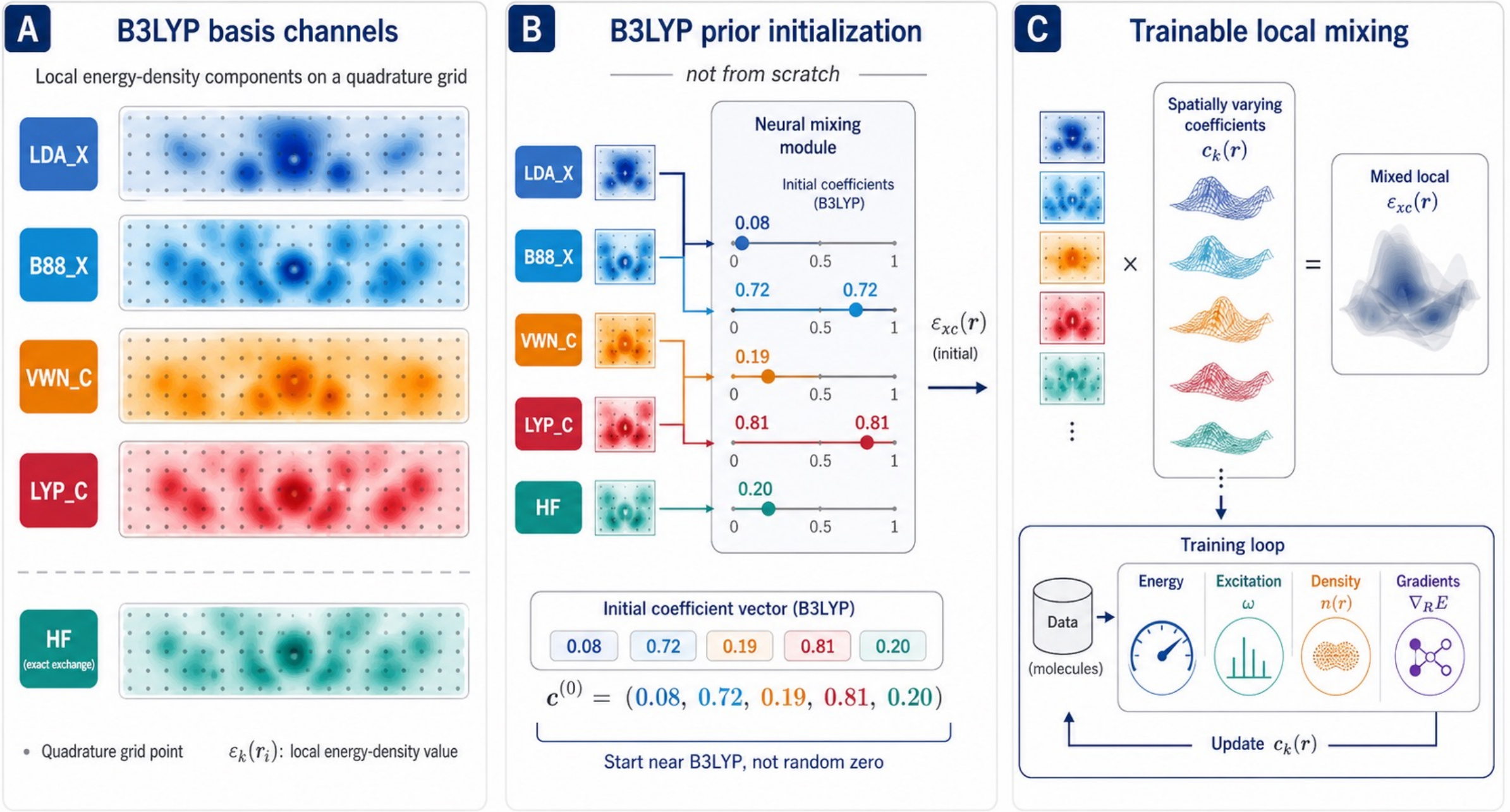


Phys. Rev. Lett. **126**, 036401

■ GradTDDFT – Training Modes in Neural Network XC Functional



■ GradTDDFT – Training Modes in Neural Network XC Functional



■ GradTDDFT – Training Modes in Neural Network XC Functional

● **Implicit SCF for H_2^+ Ground State with def2-SVP, grid level = 2**

- **N-electron system**

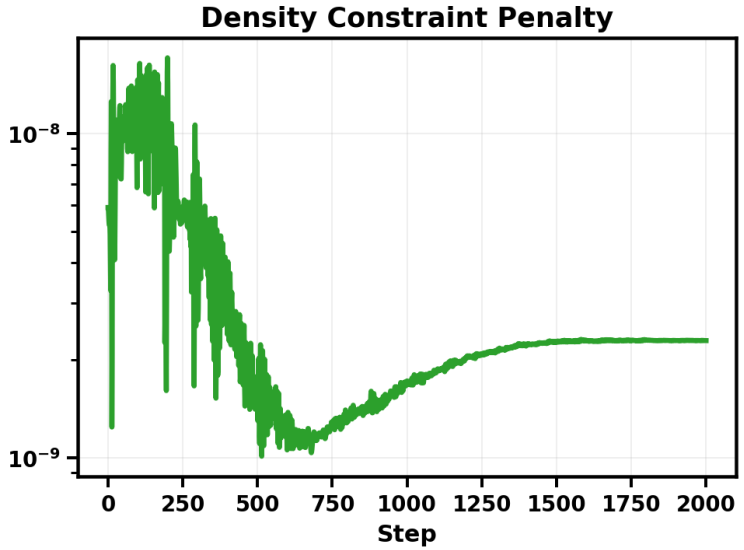
$$v_{eff}(\boldsymbol{r},\boldsymbol{R}) = v_{ext}(\boldsymbol{R}) + \int d\boldsymbol{r}' \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} + v_{xc}(\boldsymbol{r})$$

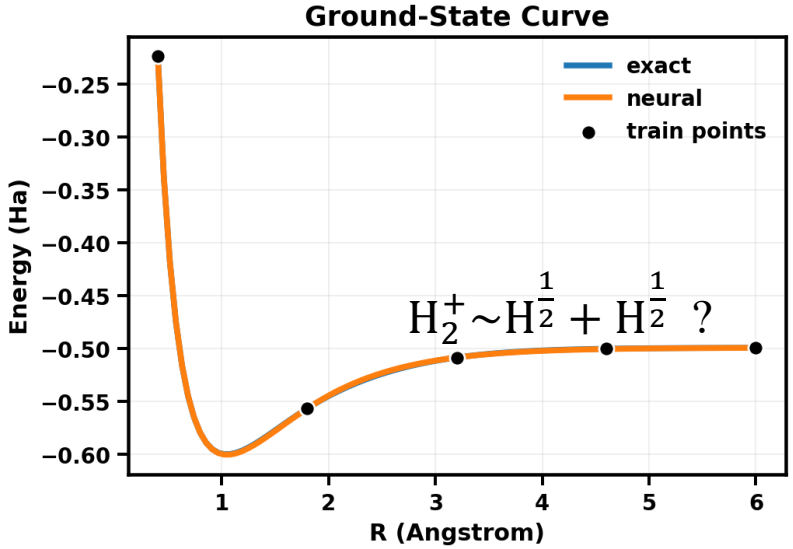
- **One-electron system($|\boldsymbol{r}| \to \infty$)**

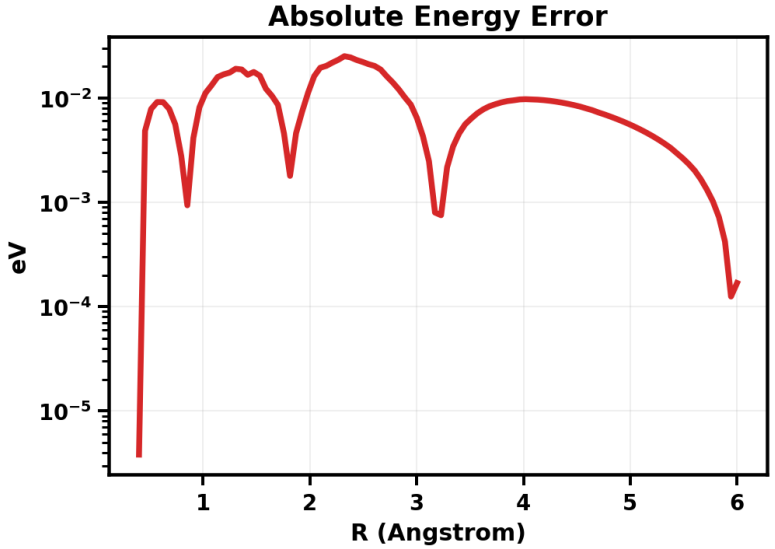
$$v_{eff}(\boldsymbol{r},\boldsymbol{R}) = v_{ext}(\boldsymbol{R}) + \frac{1}{|\boldsymbol{r}|} + v_{xc}(\boldsymbol{r})$$

- **Loss Function**

$$L = \frac{1}{N}\sum|E_{NN} - E_{FCI}| + \frac{1}{N^2}\sum(E_{NN} - E_{FCI})^2 + \frac{1}{N^2}\sum(\rho_{NN} - \rho_{FCI})^2$$







■ GradTDDFT – Training Modes in Neural Network XC Functional

● Implicit SCF for H_2 Ground State with def2-SVP, grid level = 2

- **Static Correlation Error**

  σ_g^2 **Binding Orbital**

  σ_u^2 **Antibinding Orbital**

  - **MCSCF**

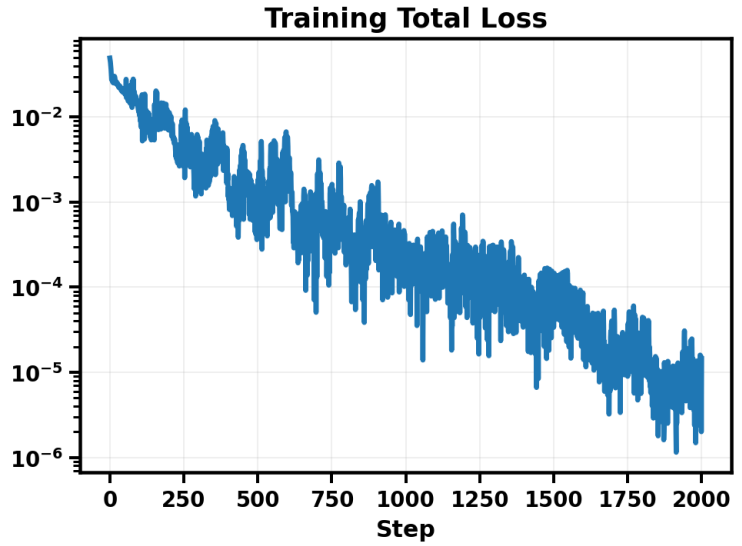
    $\psi = c_u\phi(\sigma_g^2) + c_u\phi(\sigma_u^2)$

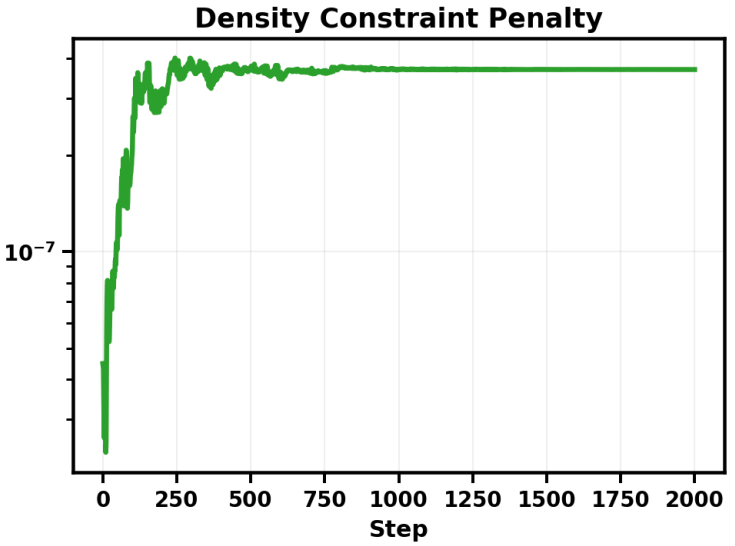
  - **Hartree-Fock**

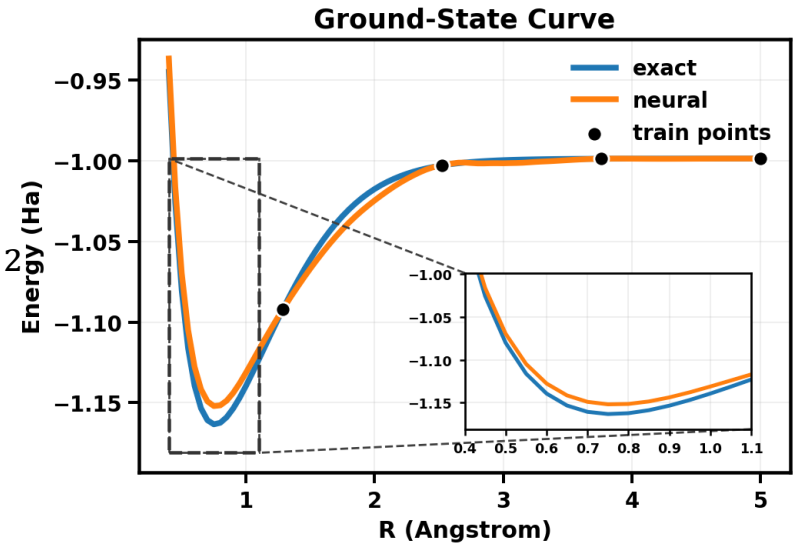
    $\psi_1 = \phi(\sigma_g^2) \quad \psi_2 = \phi(\sigma_u^2)$

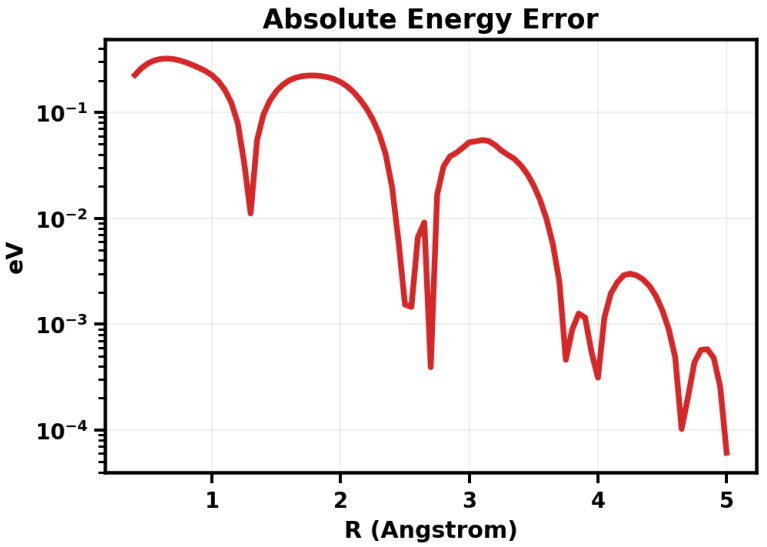
- **Loss Function**

$$L = \frac{1}{N}\sum|E_{NN} - E_{FCI}| + \frac{1}{N^2}\sum(E_{NN} - E_{FCI})^2 + \frac{1}{N^2}\sum(\rho_{NN} - \rho_{FCI})^2$$



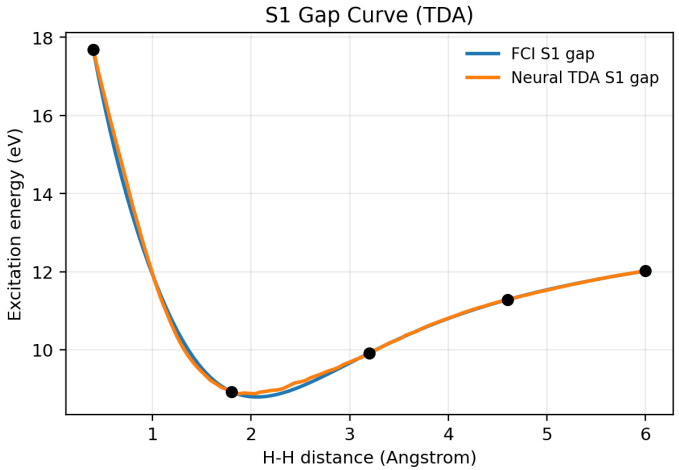


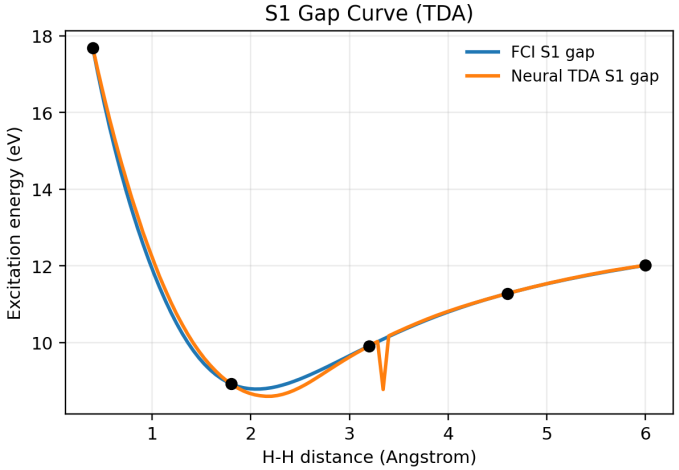


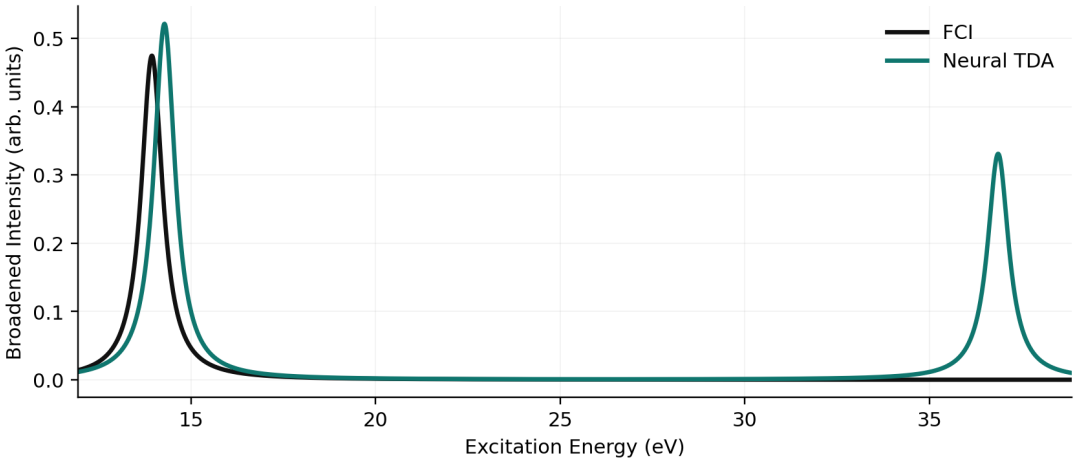


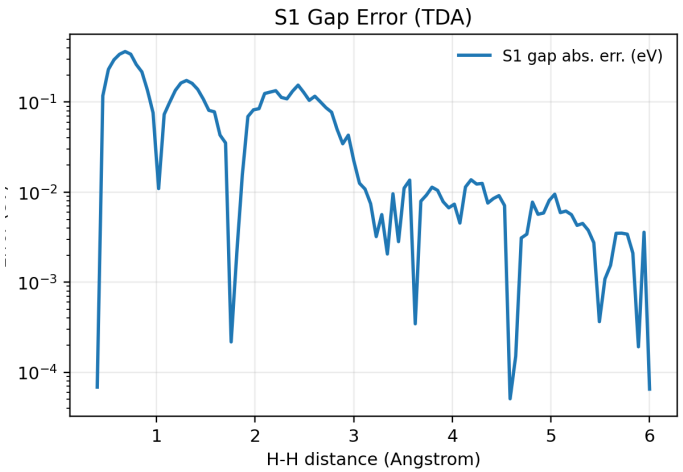
■ GradTDDFT – Training Modes in Neural Network XC Functional

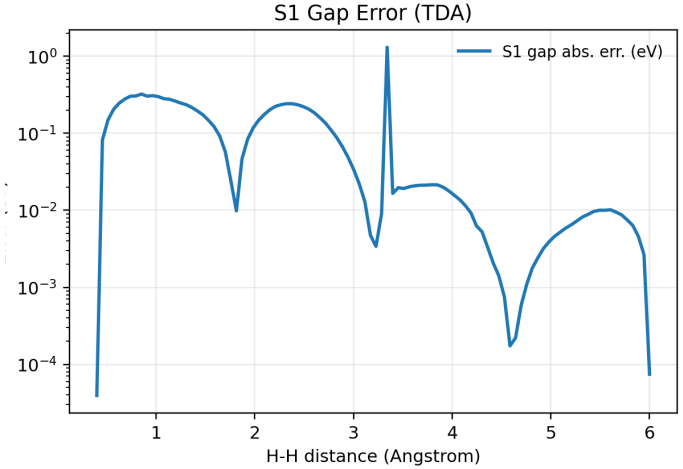
- Implicit SCF +Explicit TDA for S_1 of H_2 with def2-SVP, grid level = 2

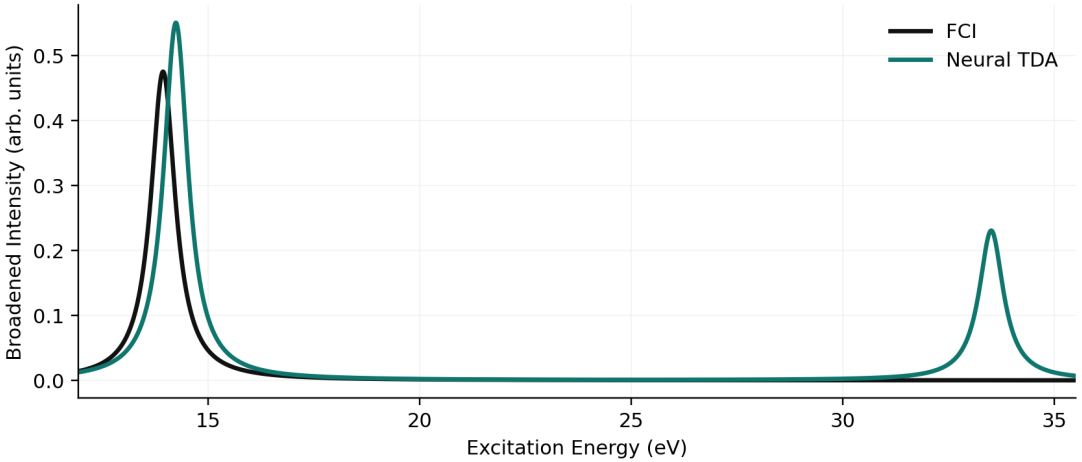






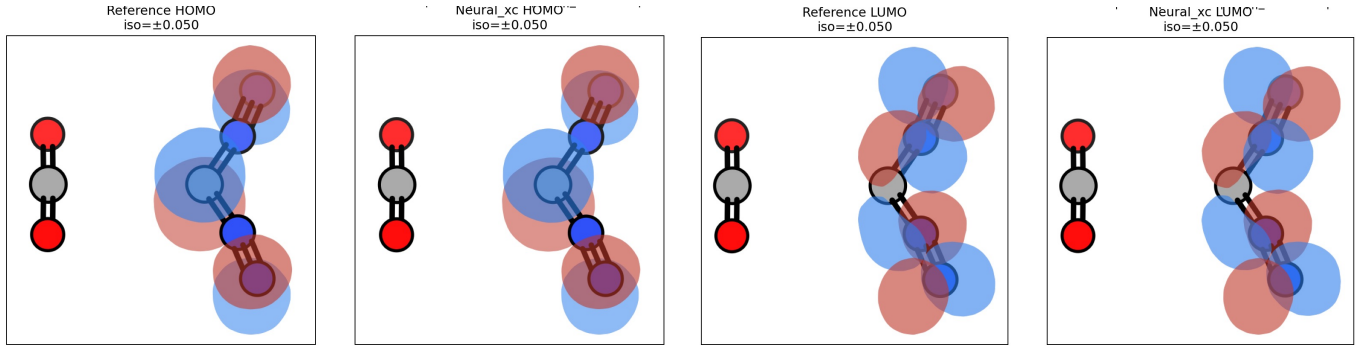


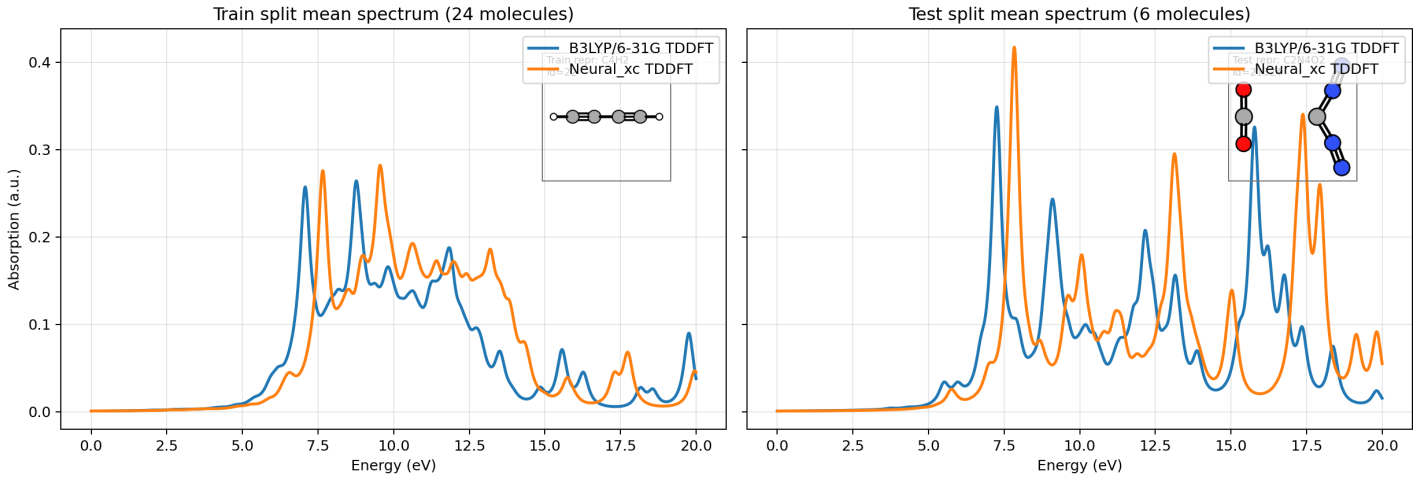


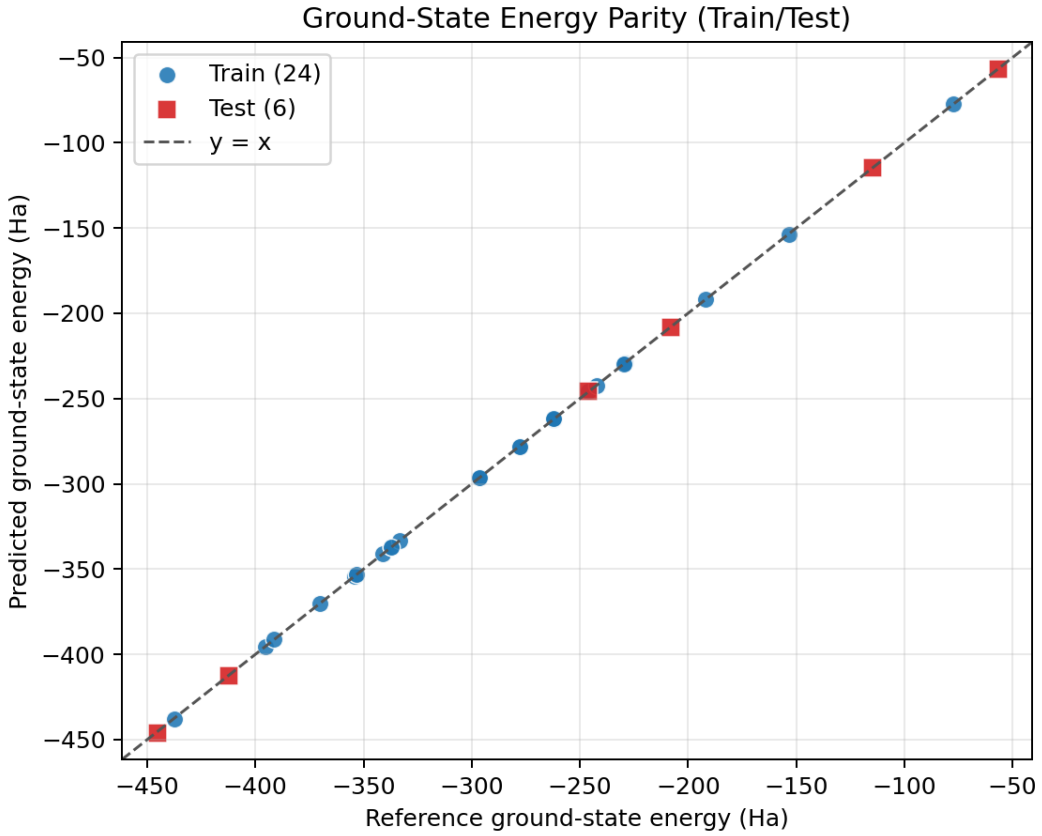


■ GradTDDFT – Training Modes in Neural Network XC Functional

- **Fixed Density for Ground State-QM9**

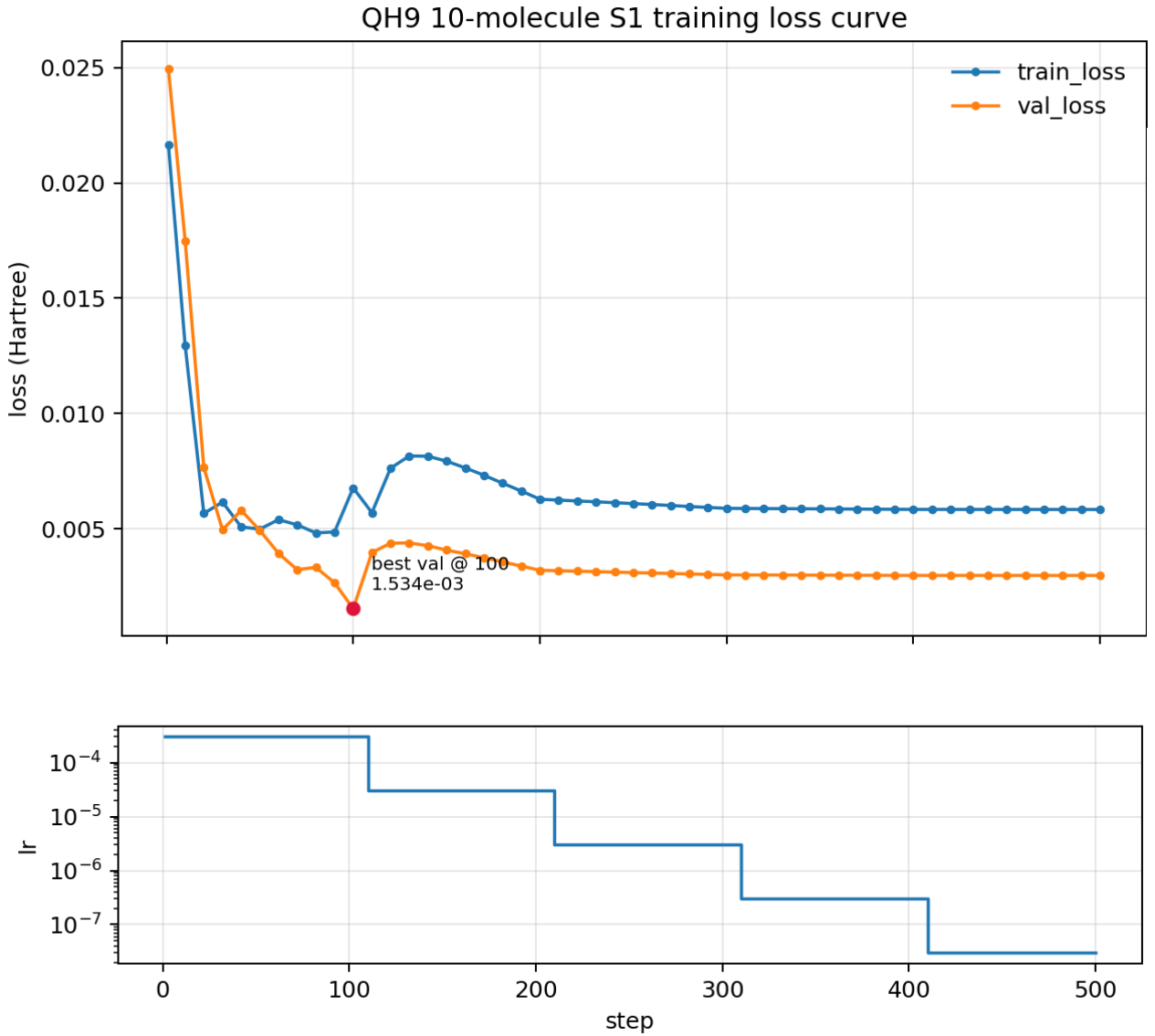


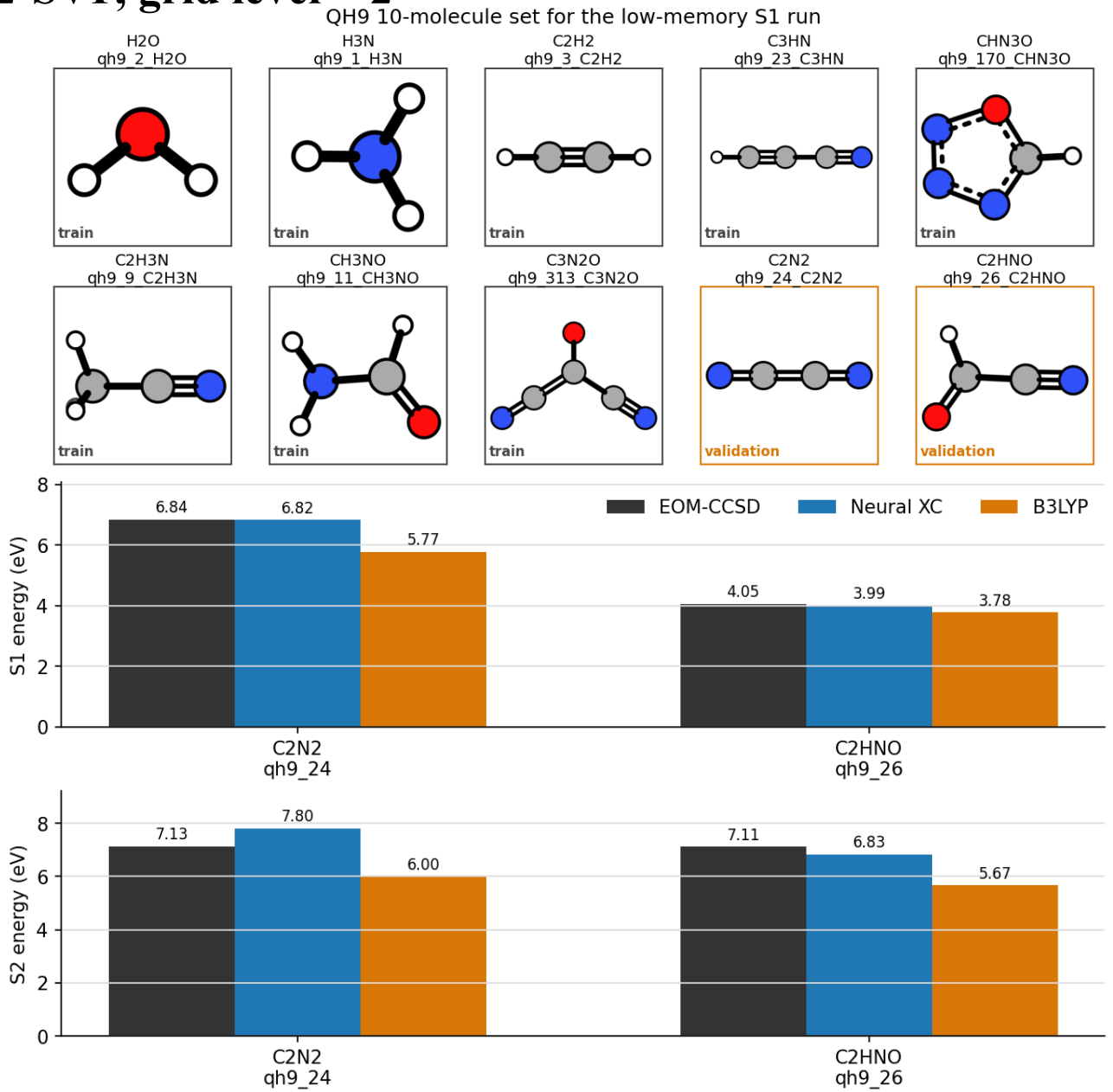




■ GradTDDFT – Training Modes in Neural Network XC Functional

- **Implicit SCF +Explicit TDA for S_1 of QM9 with def2-SVP, grid level = 2**





■ TD-GradDFT – Failure of Full Absorption Spectrum Calculation

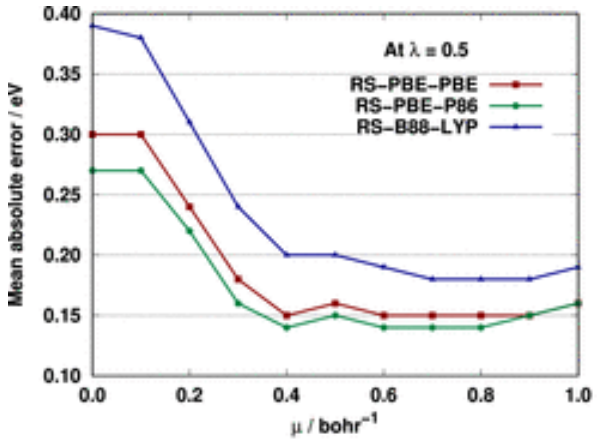
- **TDDFT is not a rigorous theorem for excited state wave function**

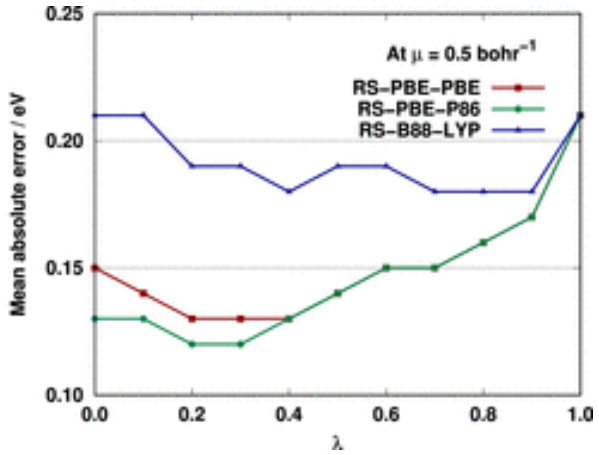
$$\psi_{ex}^{CIS} = \phi_0 + \sum_{ai} x_{ai}\phi_a^i$$

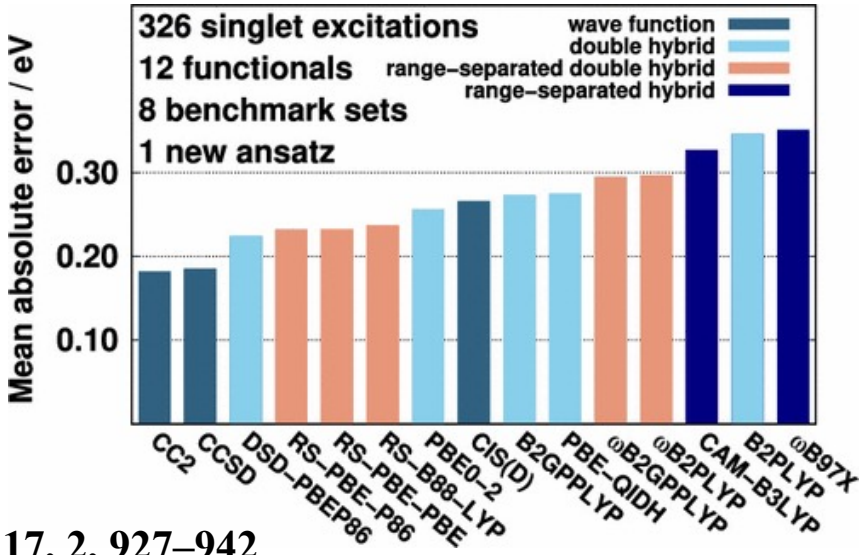
$$\psi_{ex}^{FCI} = \phi_0 + \sum_{ai} c_{ai}\phi_a^i + \sum_{aibj} c_{aibj}\phi_{ab}^{ij} + \sum_{aibjck} c_{aibjck}\phi_{abc}^{ijk} + \cdots$$

- **No Ground Truth has a good performance both on ground state and excited state for complex electron configuration**

But !





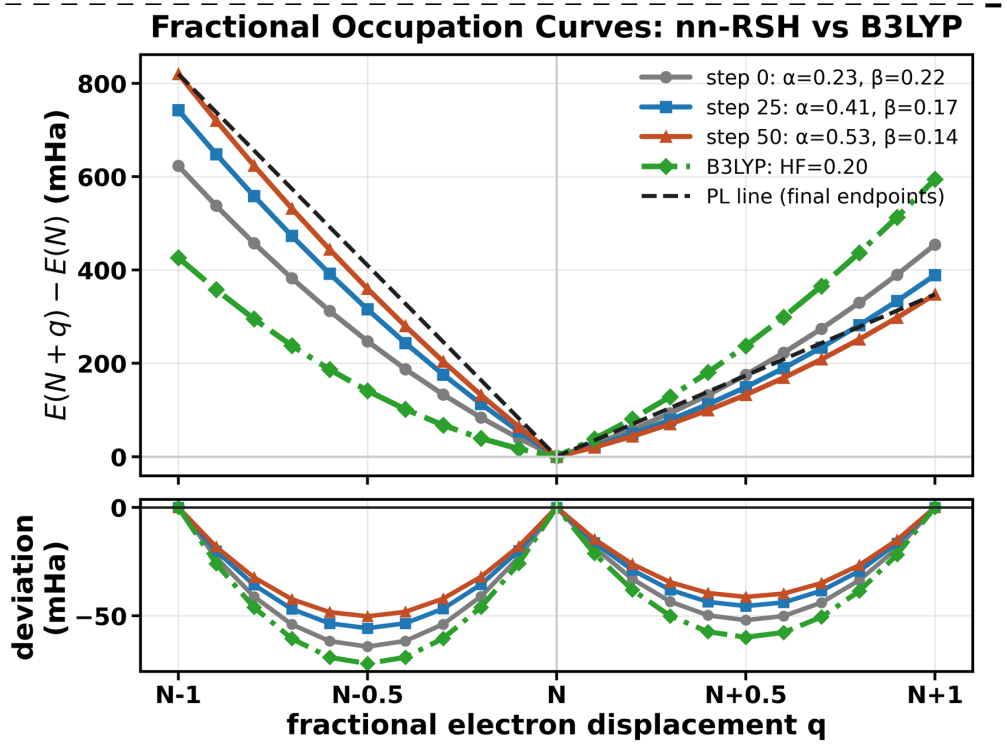


J. Chem. Theory Comput. 2021, 17, 2, 927–942

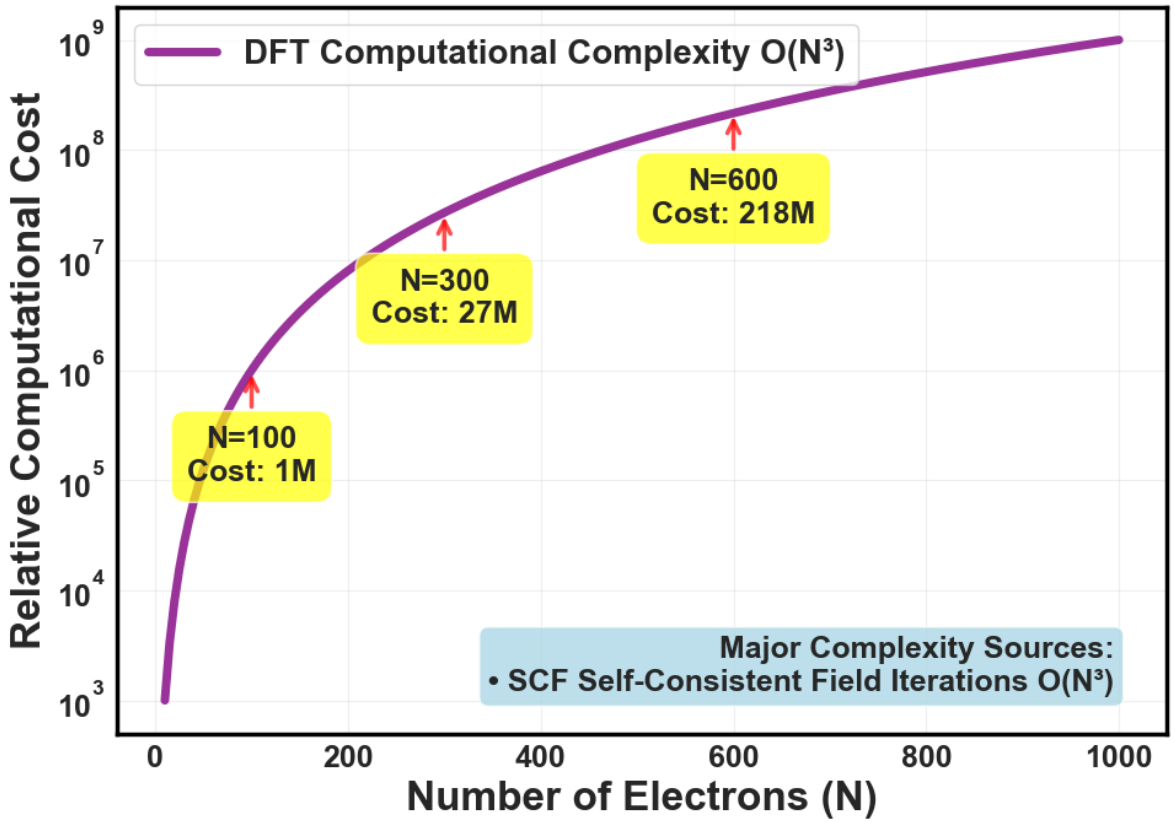
■ GradTDDFT – Functional and Outlook

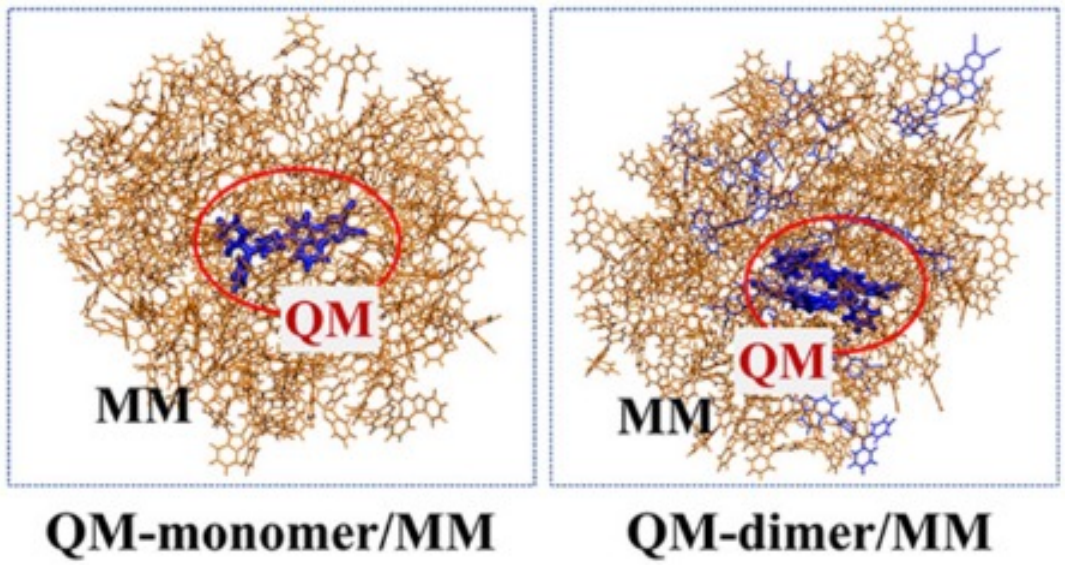
- **Functional**
  - Differential SCF code with three modes and differential LR-TDDFT code
  - HF and PT2 local energy density hybrid and CIS(D) based TDDFT for Double Hybrid Functional
  - GPU/CPU Integral Modules for DFT/TD-DFT
  - General XC Functional Library based on JAX-XC and General Base Set Library form PySCF
  - Fractional charge evaluation and differential unrestricted SCF code

- **Outlook**
  - Lower GPU Memory Usage for Large Systems!(Hard For Most Differentiable DFT)
  - Automatic Differentiation for Geometry Optimization and Response Properties of molecules
  - General Neural Network Designing, Training, Evaluating Framework for Quantum Chemistry-GradSCF



■ Equivariance in Neural Network – Covariance in DFT Hamiltonian





Phys Chem Chem Phys. 2024;26(26):18418-18425

■ Equivariance in Neural Network – Covariance in DFT Hamiltonian

Kohn-Sham Density Functional Theory

$$v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\}) \Leftrightarrow \rho(\boldsymbol{x}) \Leftrightarrow \{\psi_i(\boldsymbol{x})\}$$

$$\hat{H} = \hat{H}_0 + \hat{V} \qquad \rho(\boldsymbol{x}) = \sum_i^N \psi_i^*(\boldsymbol{x})\psi_i(\boldsymbol{x})$$

$$\rho(\boldsymbol{x},\boldsymbol{x}') = \sum_i^N \psi_i^*(\boldsymbol{x})\psi_i(\boldsymbol{x}')$$

$$\left[-\frac{1}{2}\nabla^2 + v_{ext}(\boldsymbol{x},\{\boldsymbol{R}\}) + v_H(\boldsymbol{x}) + v_{xc}(\boldsymbol{x})\right]\psi_i(\boldsymbol{x}) = \varepsilon_i\psi_i(\boldsymbol{x})$$

Phys. Rev. B 136, 864 (1964).

Atomic Basis Function

Equivariance in Atomic Basis Function

$$\chi_j(\boldsymbol{x}) = R_{nl}(r)Y_{ml}(\theta)$$

$$Y_{ml}(\theta') = \sum_{m'=-l} D^l_{m'm}Y_{m'l}(\theta)$$

$$\chi_j(\boldsymbol{R}\cdot\boldsymbol{x}) = \boldsymbol{D}(\boldsymbol{R})\chi_j(\boldsymbol{x})$$

Hamiltonian Matrix in Atomic Basis Function

$$\psi_i(\boldsymbol{x}) = \sum_a C_{ai}\chi_a(\boldsymbol{x}) \qquad HC = \varepsilon SC$$

$$H = H_{ij} = \sum_{ab} C^*_{ai}C_{bj}\,\chi_i^*(\boldsymbol{x})\hat{H}\chi_j(\boldsymbol{x})$$

Covariance in DFT Hamiltonian Matrix

$$H' = H'_{ij} = \sum_{ab} C^*_{ai}C_{bj}\,\chi_i^*(\boldsymbol{R}\cdot\boldsymbol{x})\hat{H}\chi_j(\boldsymbol{R}\cdot\boldsymbol{x})$$

$$H' = D^{\boldsymbol{T}}(\boldsymbol{R})HD(\boldsymbol{R})$$

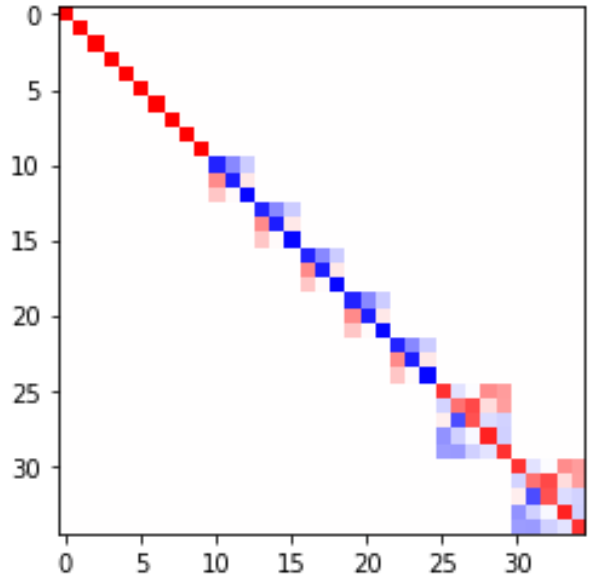
■ Equivariance in Neural Network – Rotation Matrix in Winger-D Representation

Rotation Matrix in Euler Representation

$$\mathcal{M}(\alpha,\beta,\gamma) = \mathcal{R}_z(\alpha)\mathcal{R}_x(\beta)\mathcal{R}_z(\gamma)$$

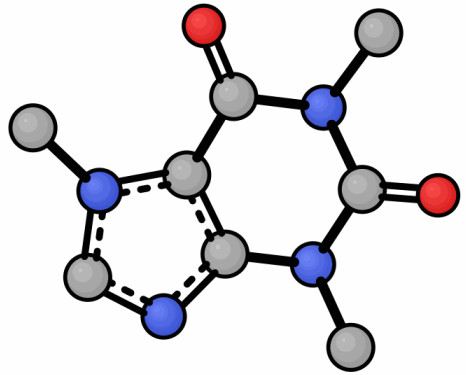
$$\begin{matrix} \cos\alpha\cos\gamma - \cos\beta\sin\alpha\sin\gamma & -\cos\beta\cos\gamma\sin\alpha - \cos\alpha\sin\gamma & \sin\alpha\sin\beta \\ = \cos\gamma\sin\alpha + \cos\alpha\cos\beta\sin\gamma & \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\sin\beta \\ \sin\beta\sin\gamma & \cos\gamma\sin\beta & \cos\beta \end{matrix}$$

Rotation Matrix in Winger-D Representation



Irreducible representation

10x0e + 5x1o + 2x2e



10 Even Scalers, 5 Odd Vectors, 2 Even Second-Order Tensors

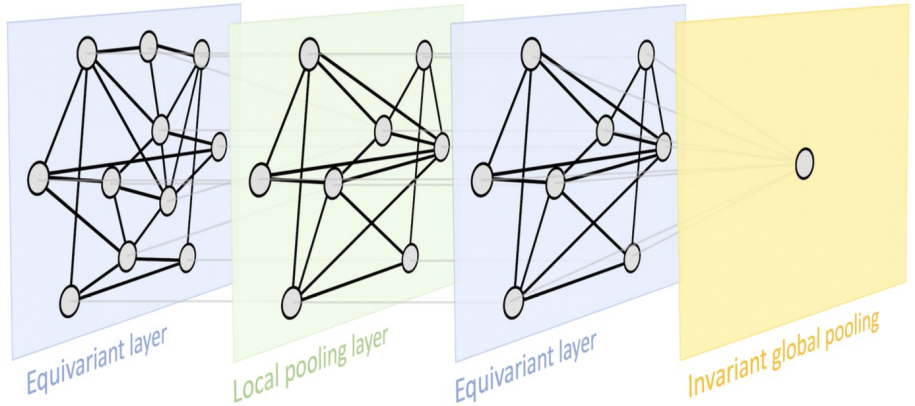
$$D^l = 10D^0 \oplus 5D^1 \oplus 2D^3$$

■ Equivariance in Neural Network – Tensor Field Network

- **Tensor Product in SO(3) Representation-Winger-Eckert Theorem**

$$\nu_1^{l_1} \otimes \nu_1^{l_1} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C_{l_1,m_1,l_2m_2}^{l_3m_3} \nu_{1m_1}^{l_1} \nu_{2m_2}^{l_2}$$

$$\nu_1^{l_1} \otimes \nu_1^{l_1} = \nu^{|l_1-l_2|} \oplus \nu^{|l_1-l_2|+1} \oplus \nu^{|l_1-l_2|+2} \oplus \cdots \oplus \nu^{|l_1+l_2|}$$

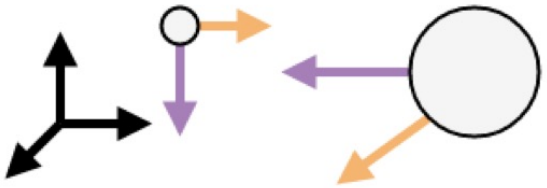


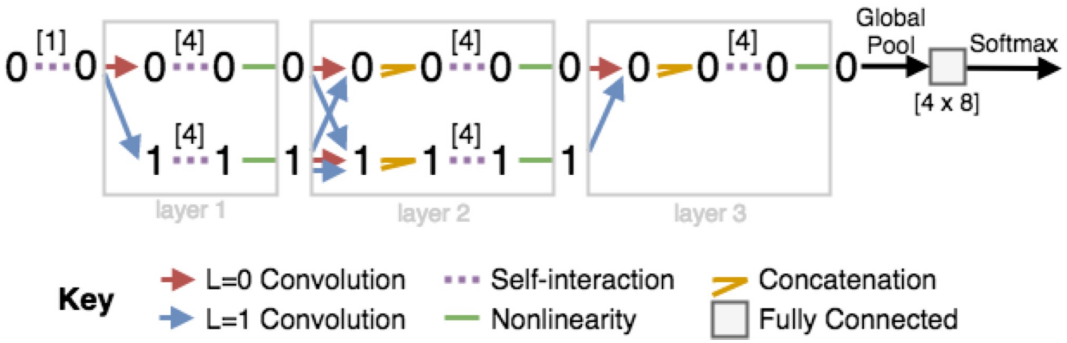
- **Massage Passing in TFN**

$$f_i' = \frac{1}{\sqrt{z}} \sum_{\mathcal{N}_i} f_j \otimes h(\| \overrightarrow{x_{ij}} \|) Y\left(\frac{\overrightarrow{x_{ij}}}{\| \overrightarrow{x_{ij}} \|}\right)$$

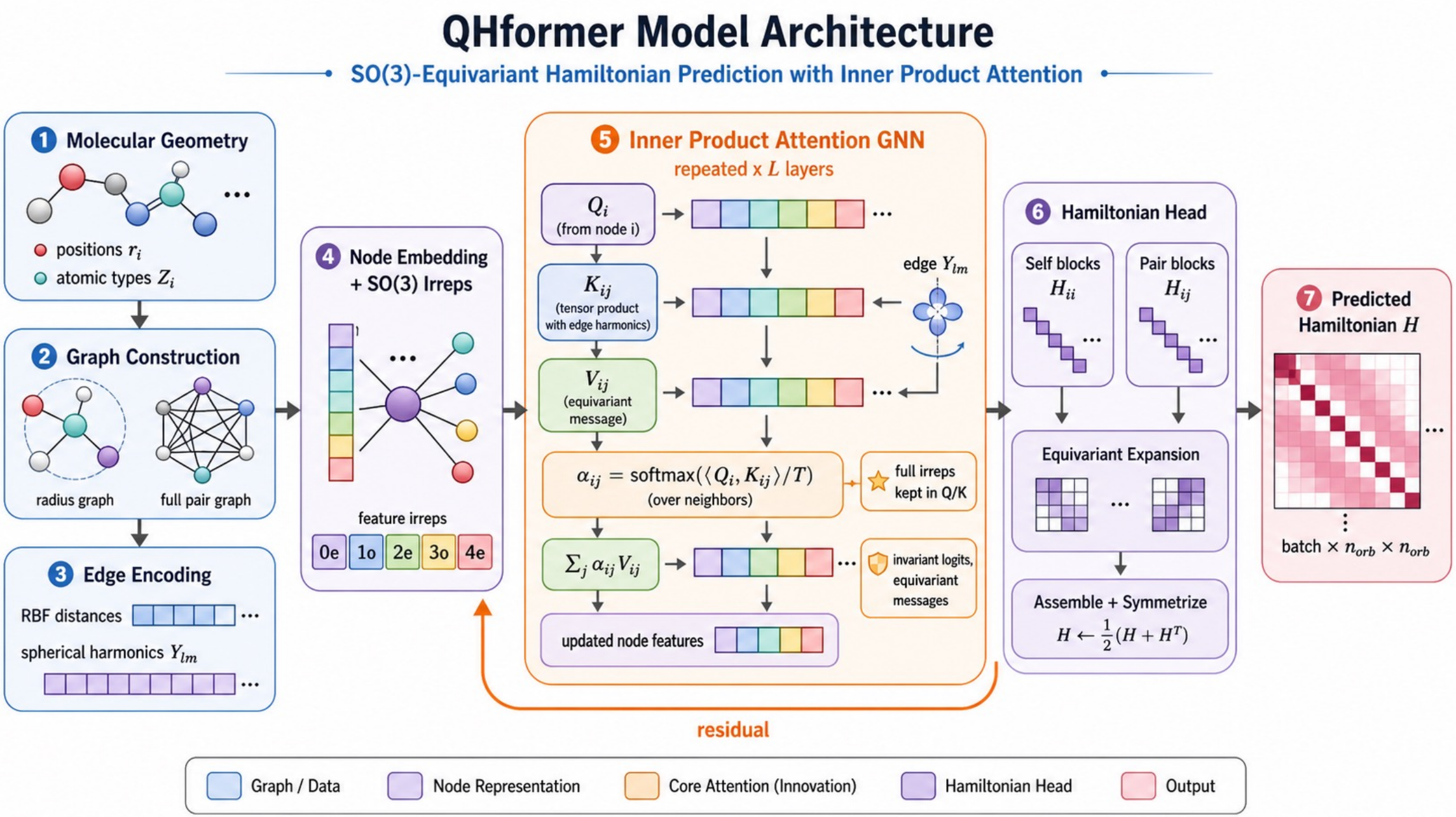
$$V_{acm}^{(l)} = \begin{matrix}\{0: & [[[\mathbf{m0}]],[[\mathbf{m1}]]],\\ 1: & [[[v0x,\ v0y,\ v0z],[a0x,\ a0y,\ a0z]],\\ & \ [[v1x,\ v1y,\ v1z],[a1x,\ a1y,\ a1z]]]\}\end{matrix}$$

1 : dictionary key, l
[ ] point index, a
[ ] channel index, c
[ ] representation index, m





■ Equivariance in Neural Network – Basic Architecture of QHformer



■ Equivariance in Neural Network – Basic Architecture of QHformer

InnerProduct Attention Mechanism

$$q_i = Linear(x_i)$$

$$k_j = TP\left(x_j, Y\left(r_{ij}\right)\right)$$

$$IP\left(q_i, k_j\right) = \Sigma_l \Sigma_m q_i^{(l,m)} \cdot k_j^{(l,m)}$$

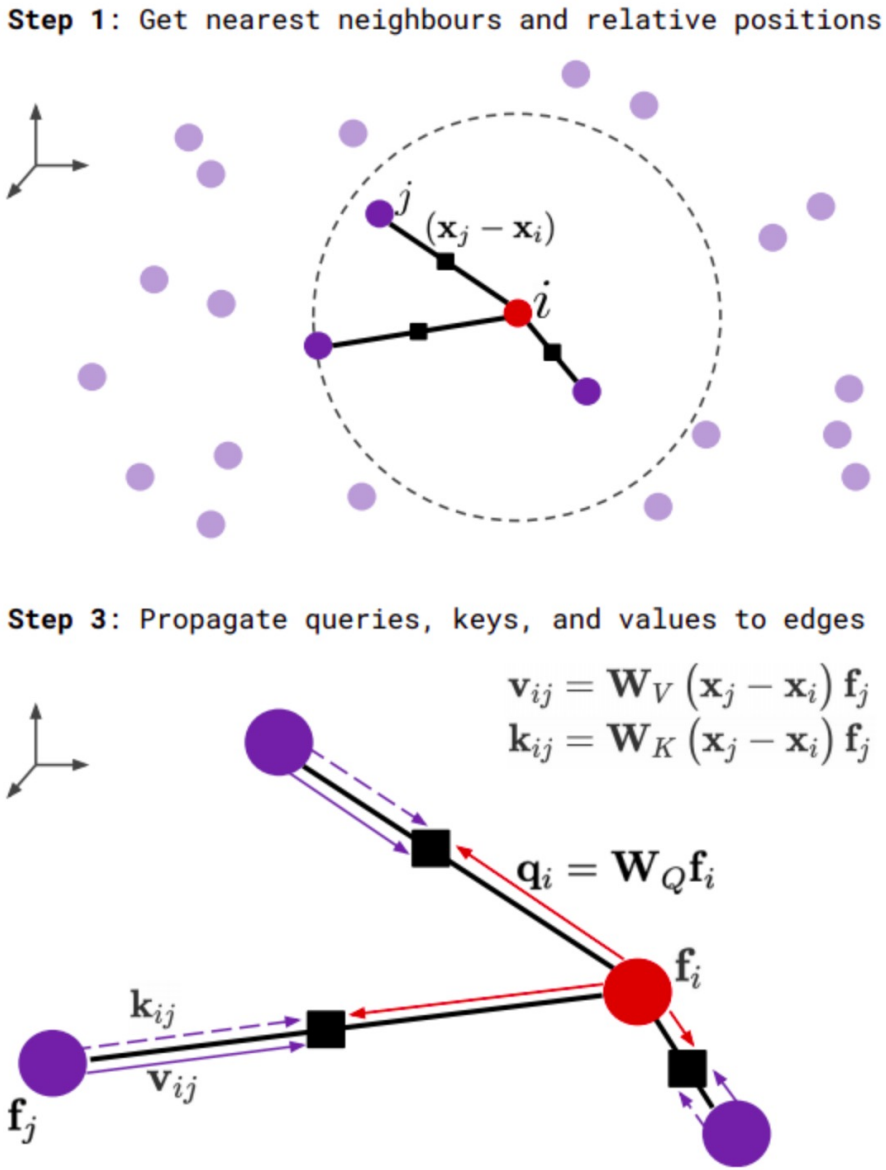
$$v_j = TP\left(x_j, Y\left(r_{ij}\right)\right)$$

$$\alpha_{ij} = softmax\left(IP\left(q_i, k_j\right)/\sqrt{d}\right)$$

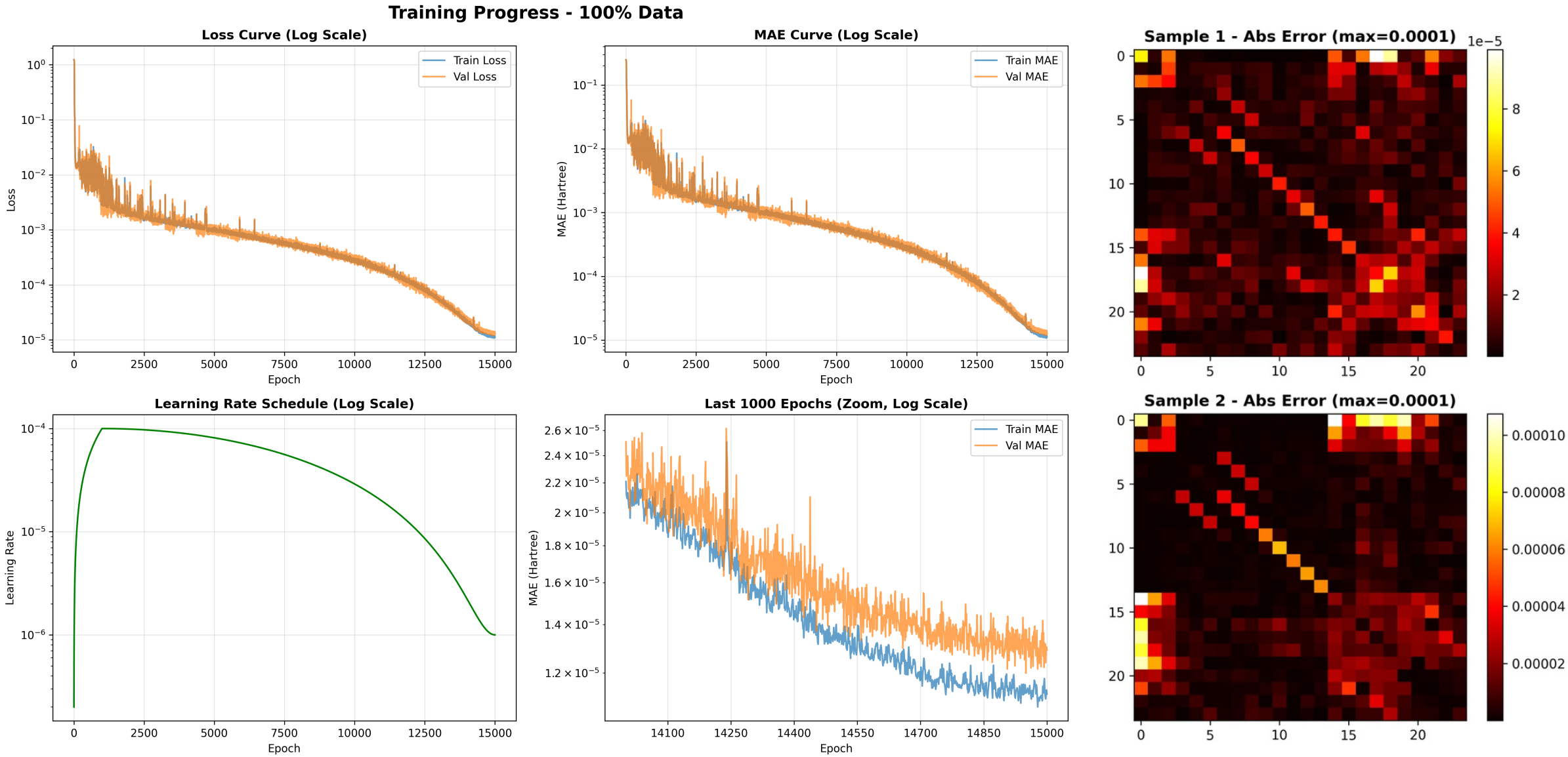
Aggregate Message with Invariance Attention Score

$$m_i = \Sigma_{j \in N(i)} \alpha_{ij} \cdot v_{ij}$$

$$x_i' = x_i + W_{out} \cdot m_i$$



■ Equivariance in Neural Network – Basic Architecture of QHformer



■ Equivariance in Neural Network – SO(2) Convolution Core in DeepTB-E3

SO(3) Convolution Filter

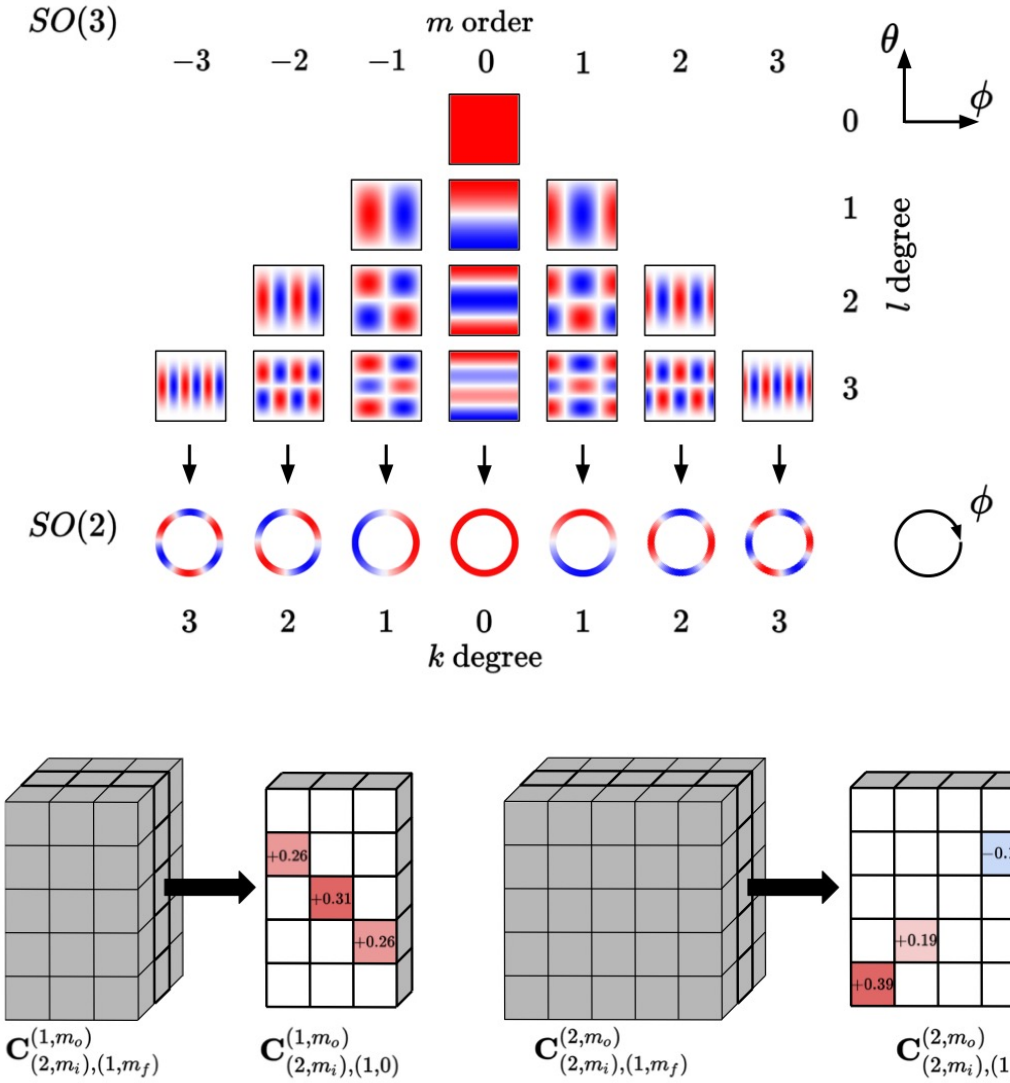
$$F_m^{l_{in}l_f}(\boldsymbol{r_{ij}}) = R^{l_{in}l_f}\left(|\boldsymbol{r_{ij}}|\right)Y_m^{l_f}\left(\frac{\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|}\right)$$

Rotation

SO(2) Convolution Filter

$$F_m^{l_{in}l_f}(\boldsymbol{r_{ij}}) = R^{l_{in}l_f}\left(|\boldsymbol{r_{ij}}|\right)Y_m^{l_f}\left(\frac{\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|}\right)$$

$$= \mathrm{D}^{-1}(\boldsymbol{R})R^{l_{in}l_f}\left(|\boldsymbol{r_{ij}}|\right)Y_m^{l_f}\left(\boldsymbol{R}\cdot\frac{\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|}\right)$$



■ Equivariant Neural Network – SO(2) Attention Mechanism in SE(3)-GAT

- **Inner-Product Attention Mechanism**

$$q_i = Linear(x_i)$$

$$k_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

$$v_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

$$IP(q_i, k_j) = \Sigma_l \Sigma_m q_i^{(l,m)} \cdot k_j^{(l,m)}$$

$$\alpha_{\mathrm{ij}} = softmax(IP(q_i, k_j)/\sqrt{d})$$

- **Message Passing**

$$m_i = \Sigma_{j \in N(i)} \alpha_{\mathrm{ij}} \cdot v_{\mathrm{ij}}$$

$$x_i' = x_i + W_{\mathrm{out}} \cdot m_i$$

- **SE(3) Head Splitting Strategy**

$$q_i^l \in \mathbb{R}^{M_l \times (2l+1)}$$

$$q_i^{l,h} \in \mathbb{R}^{\left(\frac{M_l}{H}\right) \times (2l+1)}, h = 0,1,2,3 \ldots H$$

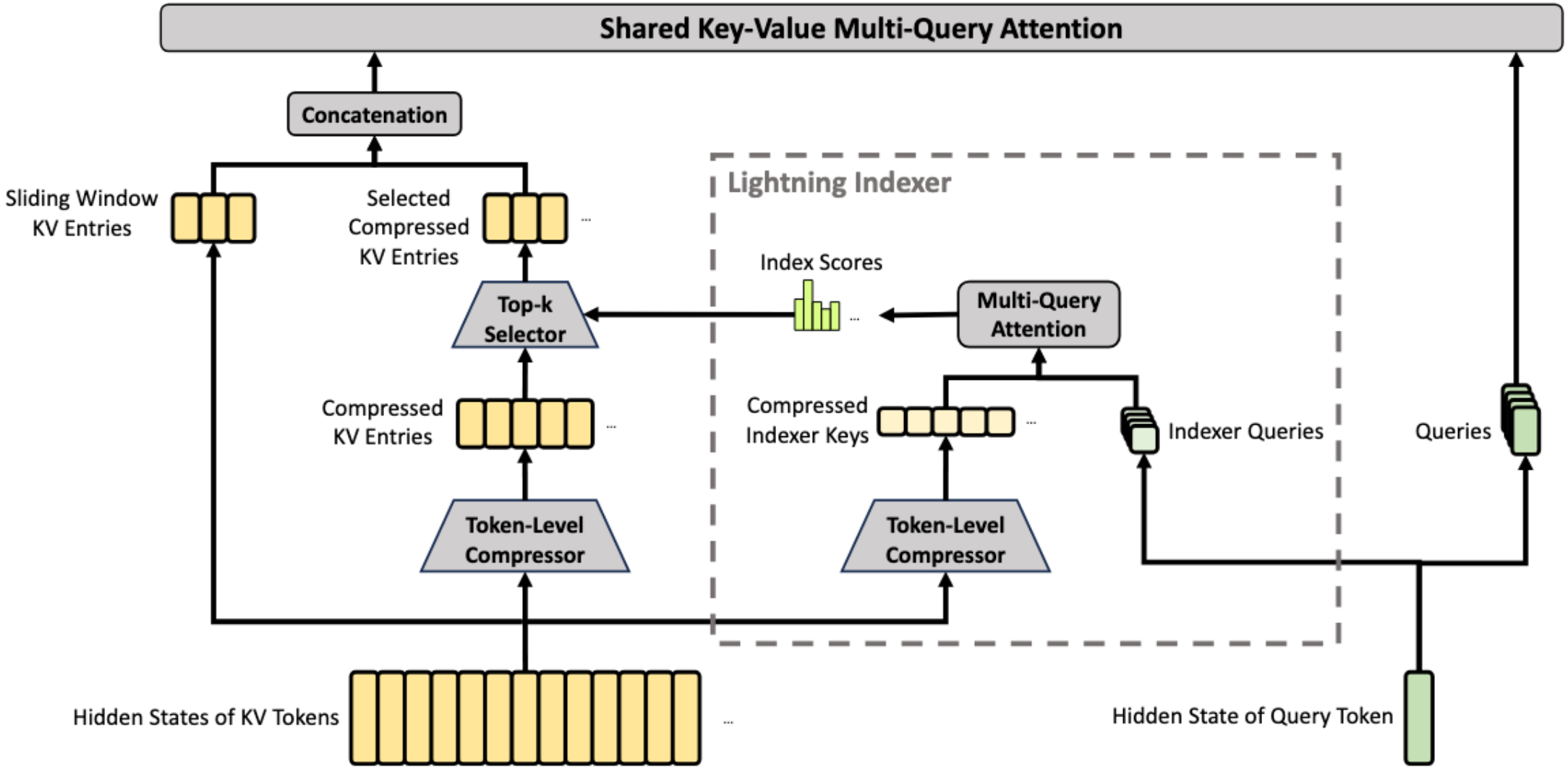
- **SO(2)-Attention Mechanism**

$$v_j = SO2Conv(x_j, r_{ij}, w_{ij}) = D^{-1}(\boldsymbol{R}) x_j Y_m^{l_f}\left(\boldsymbol{R} \cdot \frac{\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|}\right) w_{ij}$$

$$k_j = SO2Conv(x_j, r_{ij}, w_{ij}) = D^{-1}(\boldsymbol{R}) x_j Y_m^{l_f}\left(\boldsymbol{R} \cdot \frac{\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|}\right) w_{ij}$$

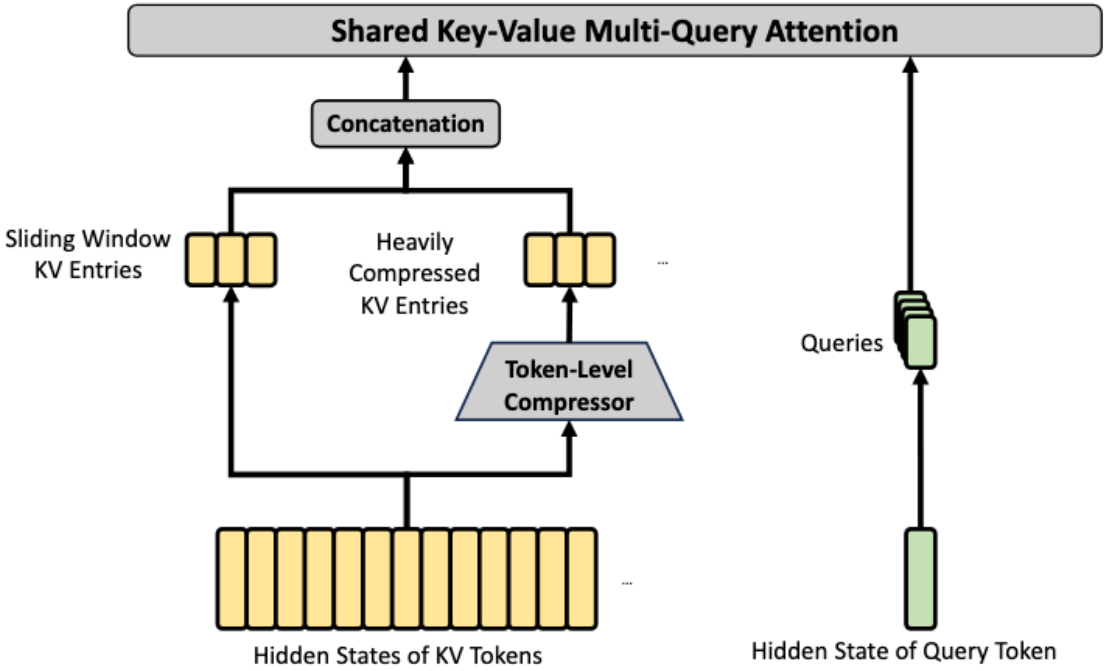
■ Equivariant Neural Network – Hybrid Attention Mode Inspired by DeepSeek V4 Pro

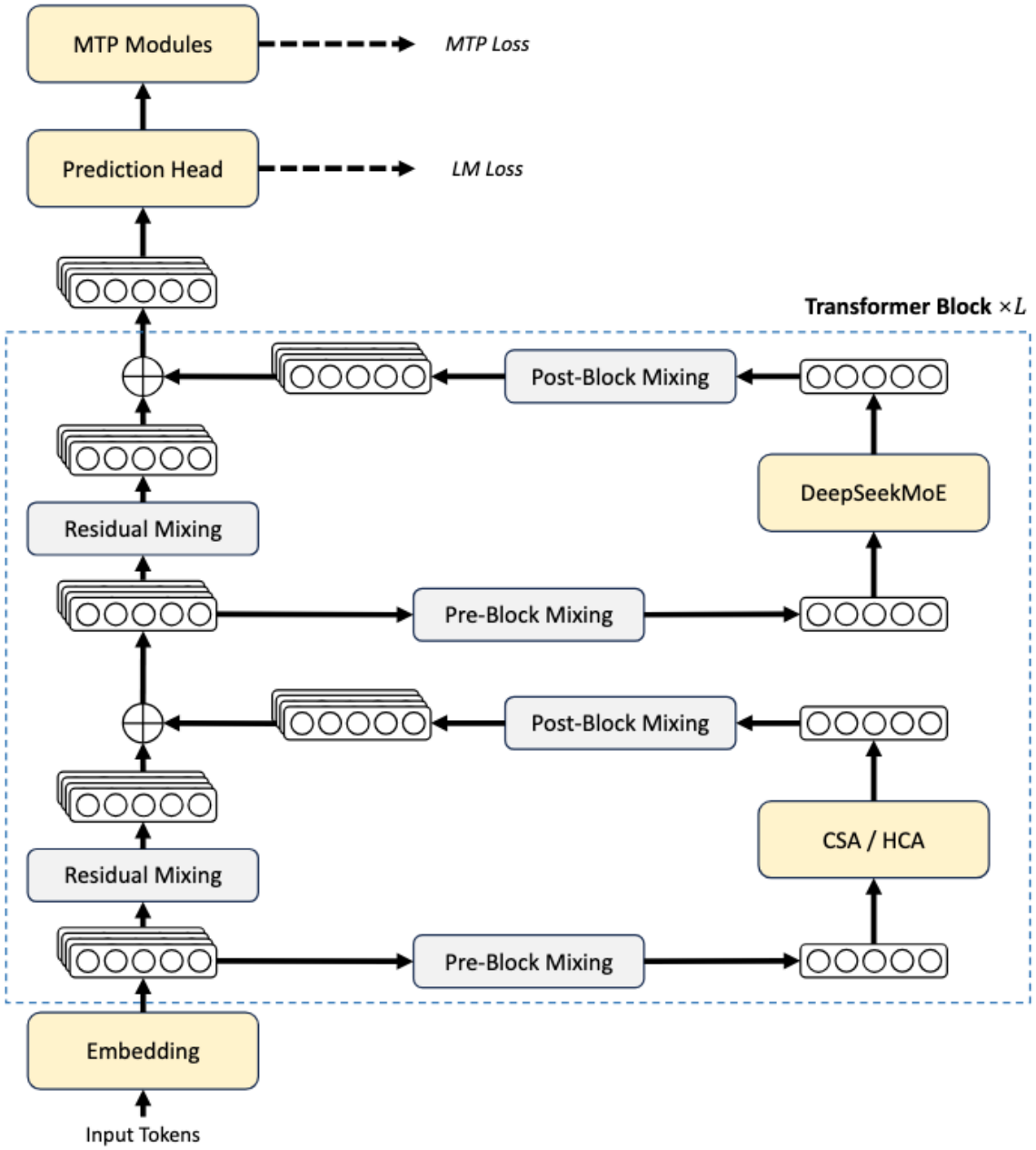
Compressed Sparse Attention(CSA)



■ Equivariant Neural Network – Hybrid Attention Mode Inspired by DeepSeek V4 Pro

Heavily Compressed Attention (HCA)





■ Equivariant Neural Network – Hybrid Attention Mode Inspired by DeepSeek V4 Pro

- **CSA in QHformer**

$$r_{ij} = I_\theta([s_{ij}, a_{ij}])$$

$$\varepsilon_i^{CSA} = TopK_{j\in N(i)}(r_{ij}; k_{CSA})$$

↓ **Only** $j \in \varepsilon_i^{CSA}$

$$k_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

$$v_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

$$\alpha_{\mathrm{ij}} = softmax(IP(q_i, k_j)/\sqrt{d})$$

$$m_i = \Sigma_{j\in\varepsilon_i^{CSA}} \alpha_{\mathrm{ij}} \cdot v_{\mathrm{ij}}$$

$$x_i' = x_i + W_{\mathrm{out}} \cdot m_i$$

- **HCA in QHformer**

$$k_{ij} \in V_{L_{max}} \quad v_{ij} \in V_{L_{max}}$$

↓ **Only** $l \le L_{max}$ **and** $j \in \varepsilon_i^{CSA}$

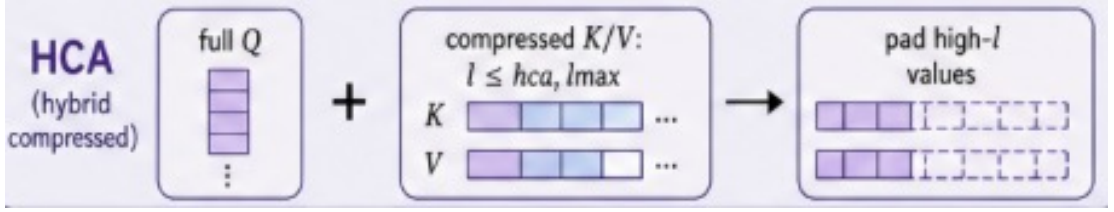
$$k_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

$$v_{ij} = SO2Conv(x_j, r_{ij}, w_{ij})$$

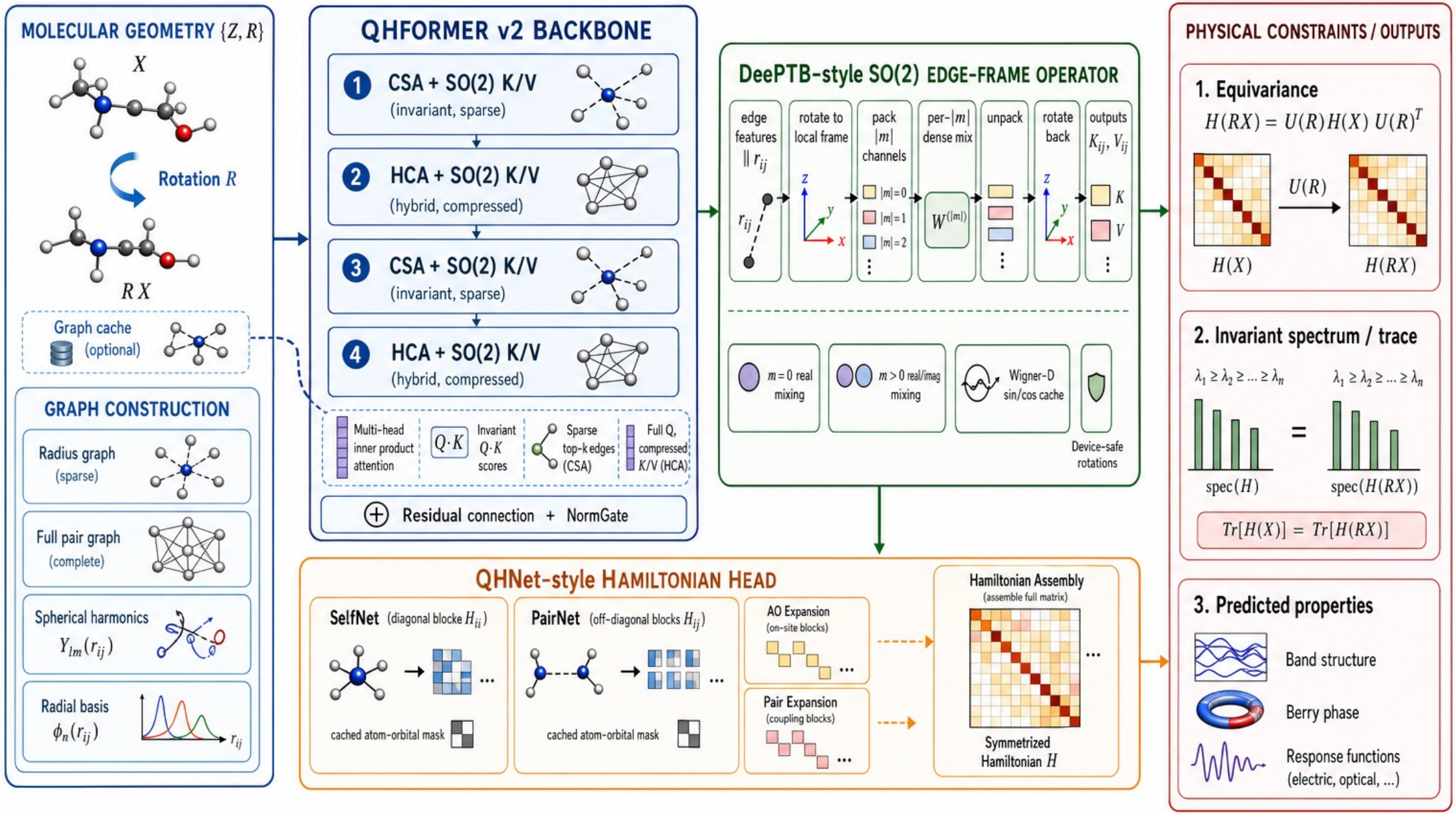
$$\alpha_{\mathrm{ij}} = softmax(IP(q_i, k_j)/\sqrt{d})$$

$$m_i = \Sigma_{j\in\varepsilon_i^{CSA}} \alpha_{\mathrm{ij}} \cdot v_{\mathrm{ij}}$$

$$x_i' = x_i + W_{\mathrm{out}} \cdot m_i$$

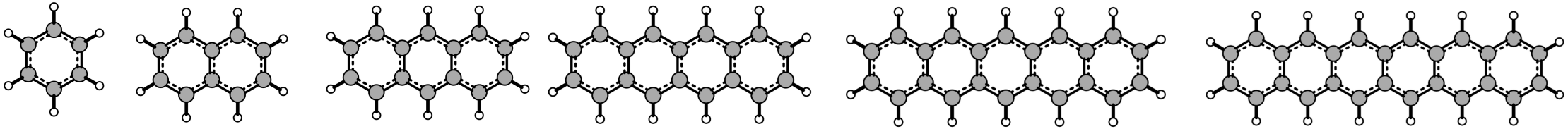


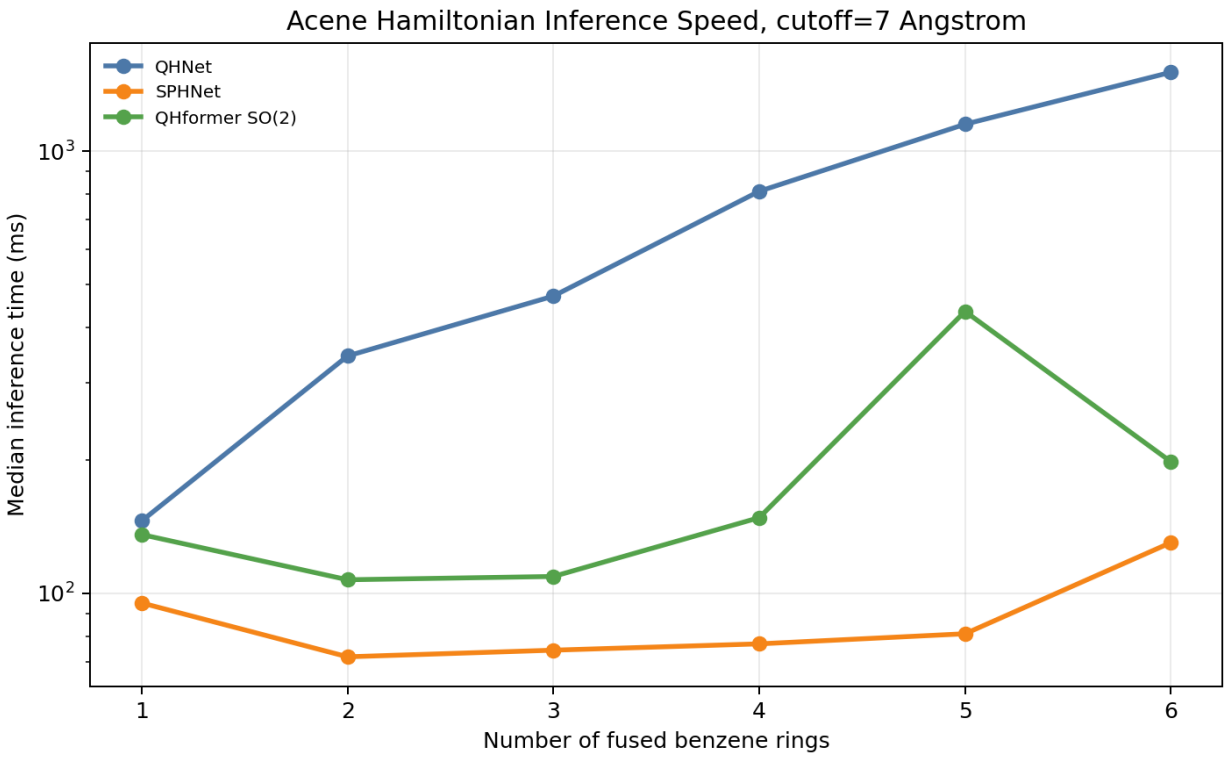
■ Equivariant Neural Network – Basic Architecture of QHfomer_v2

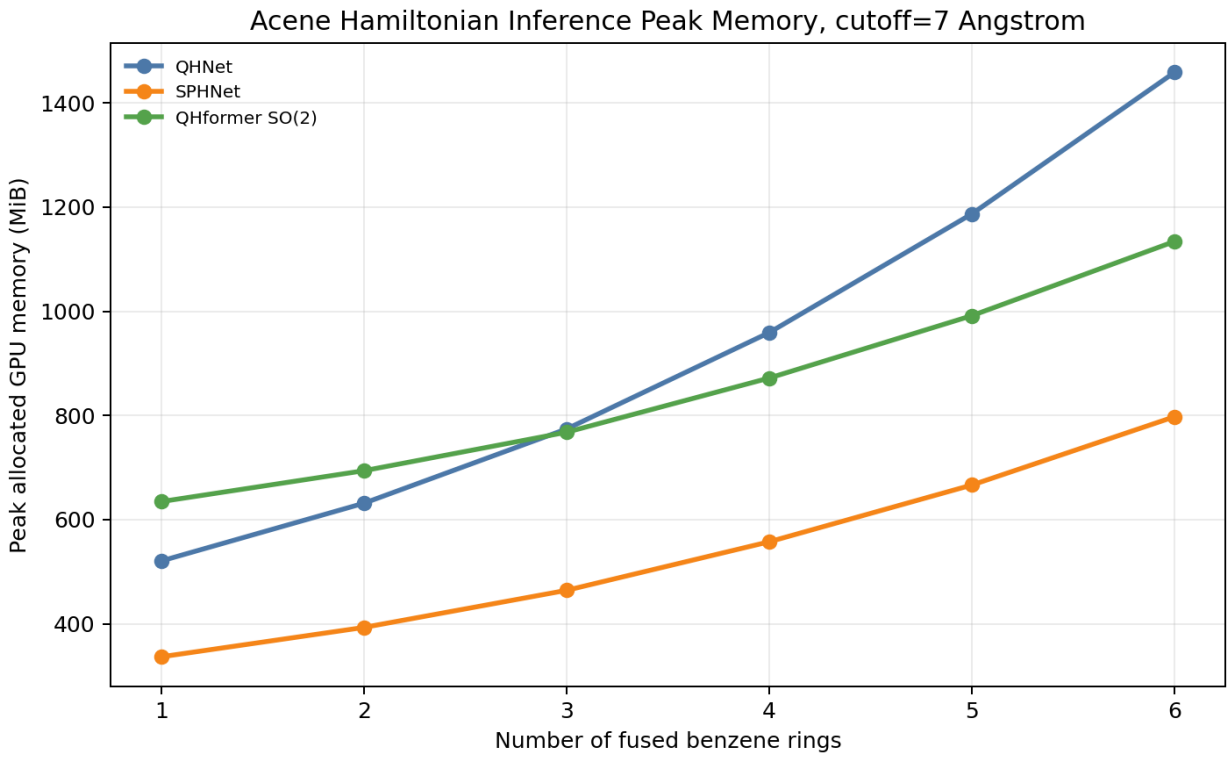


Github.com/STOKES-DOT/QHformer

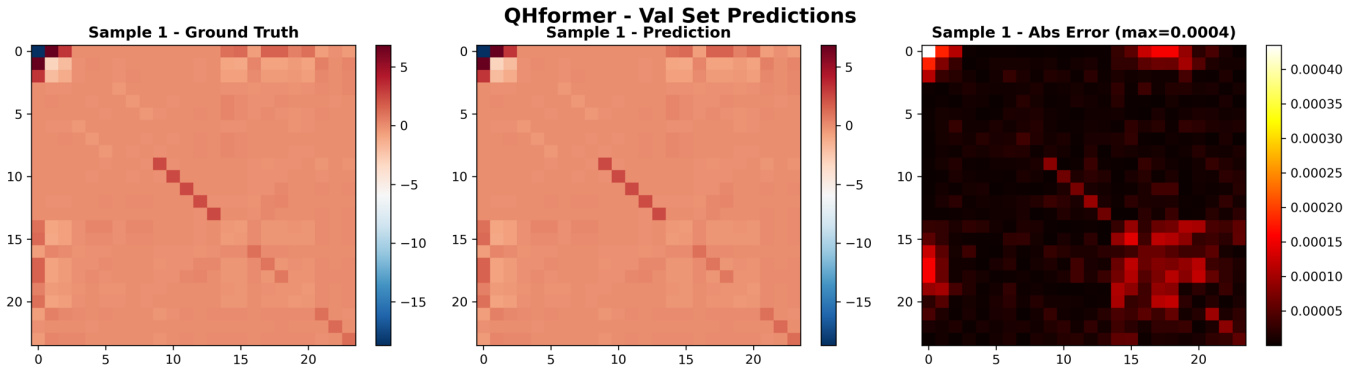
■ Equivariant Neural Network – Basic Architecture of QHfomer_v2

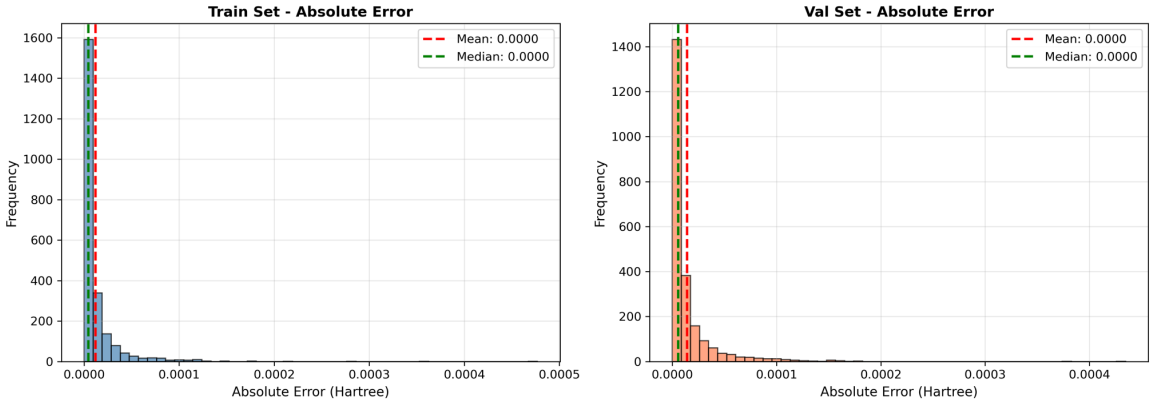






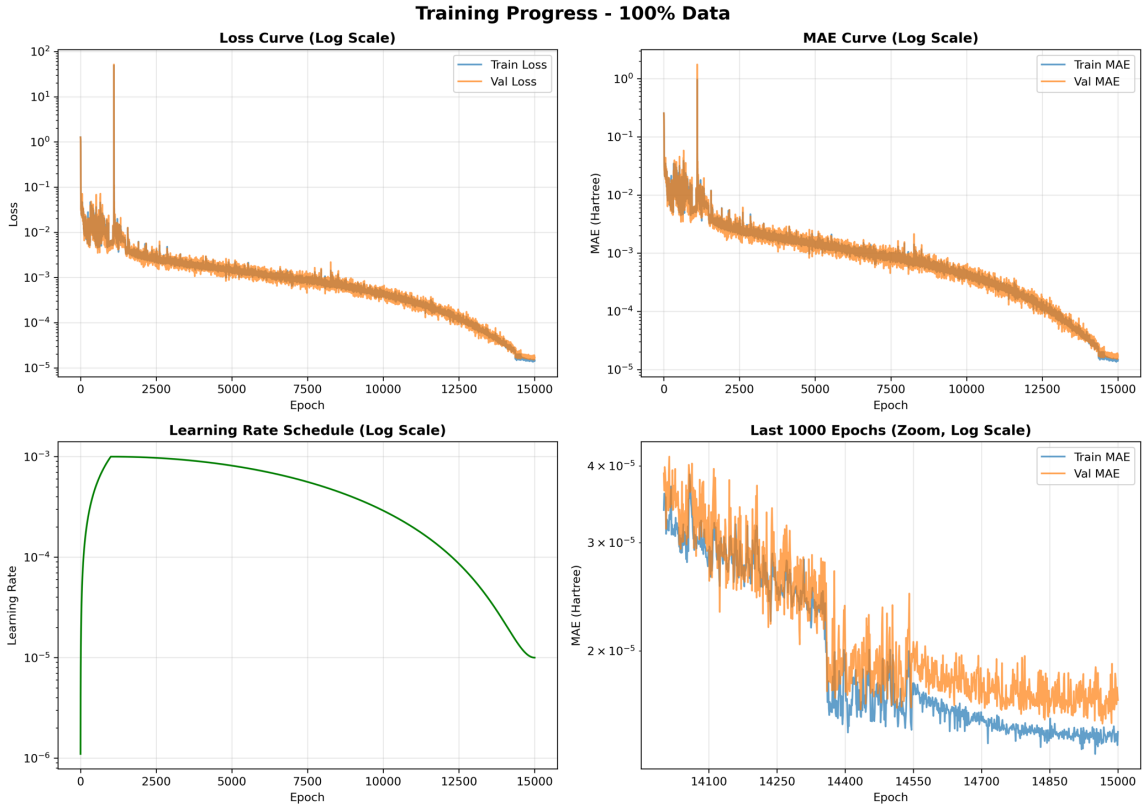
Equivariant Neural Network – Benchmark on MD17



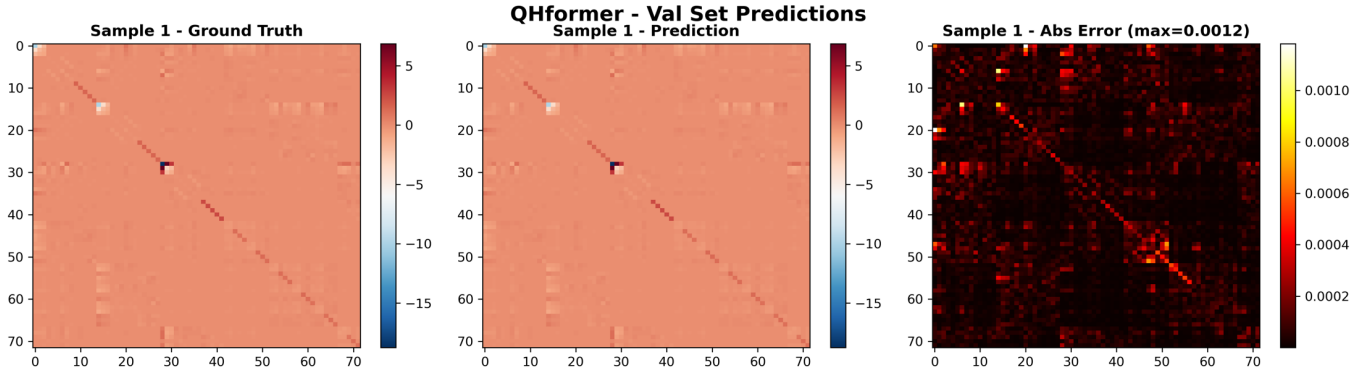


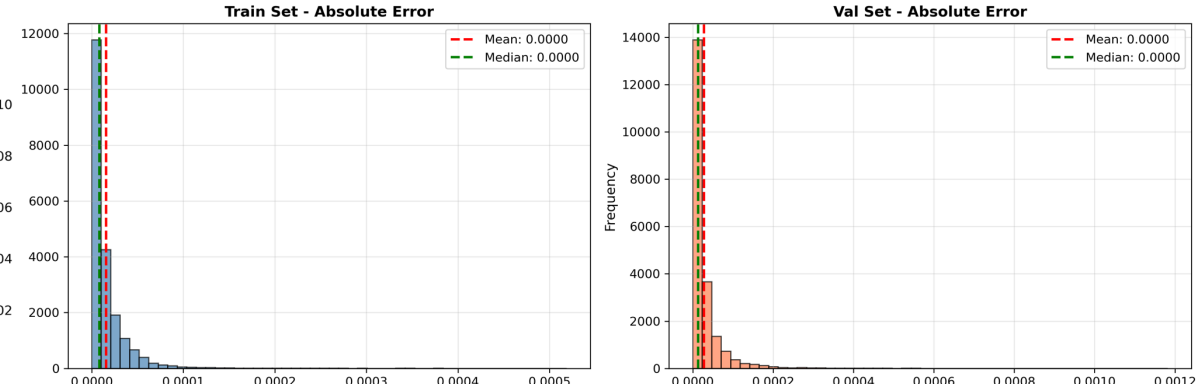
Config	Dense TP	SO(2)	Speedup	Peak memory
Operator only	2.25 ms	2.10 ms	1.07x	216 MB -> 151 MB
Full attention	10.39 ms	10.17 ms	1.02x	583 MB -> 380 MB

Variant	Runtime	Speedup vs dense TP attention	Peak memory ratio
Dense TP	11.16 ms	1.00x	100%
CSA + TP	10.53 ms	1.06x	33.1%
CSA + SO(2)	9.65 ms	1.16x	24.0%
HCA + TP	10.61 ms	1.05x	27.1%
HCA + SO(2)	9.53 ms	1.17x	24.4%



Equivariant Neural Network – Benchmark on MD17





Config	Dense TP	SO(2)	Speedup	Peak memory
Operator only	2.25 ms	2.10 ms	1.07x	216 MB -> 151 MB
Full attention	10.39 ms	10.17 ms	1.02x	583 MB -> 380 MB

Variant	Runtime	Speedup vs dense TP attention	Peak memory ratio
Dense TP	11.16 ms	1.00x	100%
CSA + TP	10.53 ms	1.06x	33.1%
CSA + SO(2)	9.65 ms	1.16x	24.0%
HCA + TP	10.61 ms	1.05x	27.1%
HCA + SO(2)	9.53 ms	1.17x	24.4%

■ Equivariant Neural Network – **Functional and Outlook**

- **Functional**

  - **A New SO(2) Attention Mechanism for Graph Transformer Inspired by DeepSeek V4 Pro**
  - **Basic SE(3)-Graph Transformer Architecture for Molecular Representation**
  - **Efficient and Accurate Hamiltonian Learning Architecture for DFT Calculation**

- **Outlook**

  - **SE(3)-Graph Transformer Architecture Training for other Equivariance-Need Tasks**
  - **SO(2) PairNet and SelfNet Modules or Better**
  - **Combine with Full Differential DFT/TDDFT Code**
  - **Combine with HamGNN, DeepH-2…(Graph Transformer based Hamiltonian Learning)**

■ DL-TDDFT – Accelerating TDDFT with CIS-like Graph Neural Network

Davidson Method Based

- **Subspace**

$V_m = [v_1, v_2, \dots, v_m] \qquad W_m = AV_m$

- **Residuals**

$H_m = V_m^T A V_m = V_m^T W_m$

$H_m y_k = \theta_k y_k, x_k = V_m y_k$

$r_k = Ax_k - \theta_k x_k$

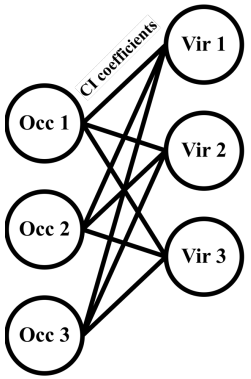
- **Expansion**

$t_k \approx (\mathrm{diag(A)} - \theta_k I)^{-1} r_k$

$V_{(m+1)} = orth([V_m, t_1, t_2, \dots])$

Graph Neural Network Based

$n_p = [C_p || \epsilon_p]$ **GAT-V2**



- **Node**

$o_i = f_{occ}(n_i) \qquad \nu_a = f_{vir}(n_a)$

- **Edge**

$e_{ai} = f_{vir}(\eta_{ai}) \qquad \eta_{ai} = [r_{ai}, \Delta\epsilon_{ai}, d_{ai}]$

- **Interaction and Transition**

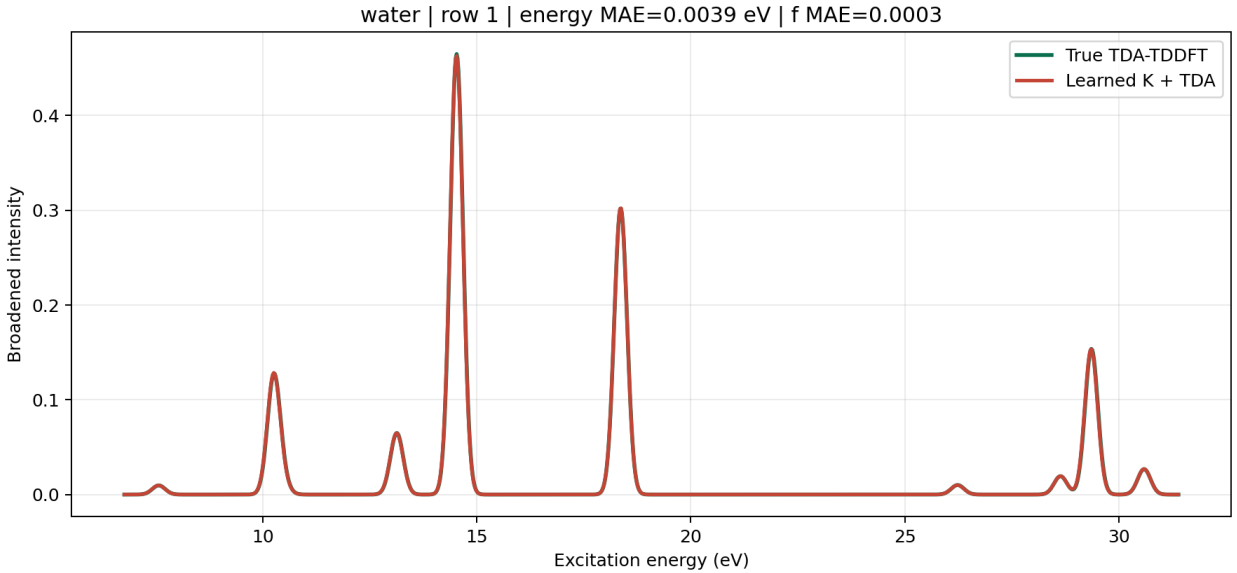
$z_{ai} = f_{int}([o_i^l \otimes v_a^l, o_i^l - v_a^l, e_{ai}^l])$

- **Pair-Pair Decoder**

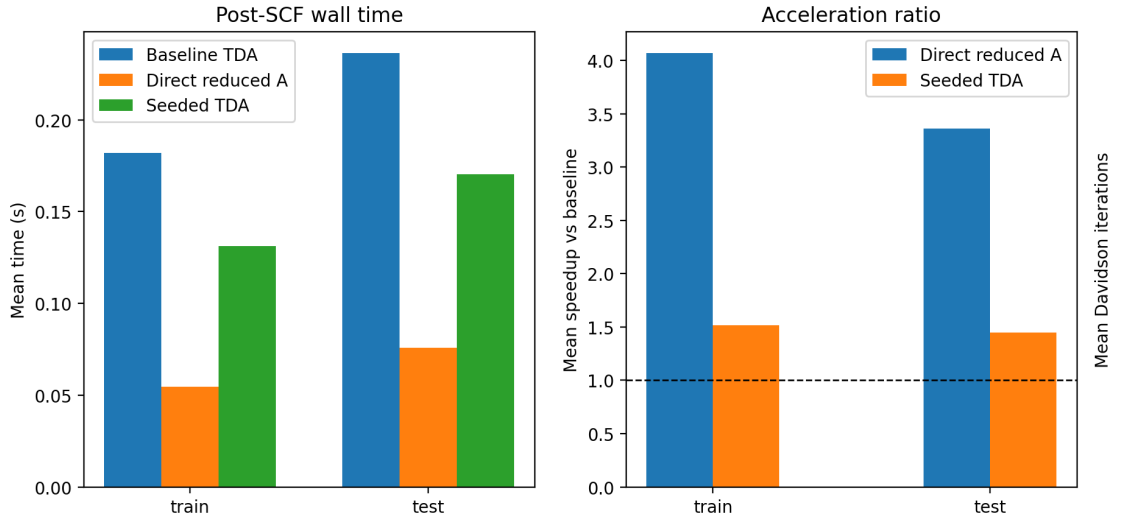
$D_{ai,bj} = f([z_{ai} \otimes z_{bj}, z_{ai} - z_{bj}, \phi_{ai,bj}])$

■ DL-TDDFT – A Test for Accelerating TDDFT

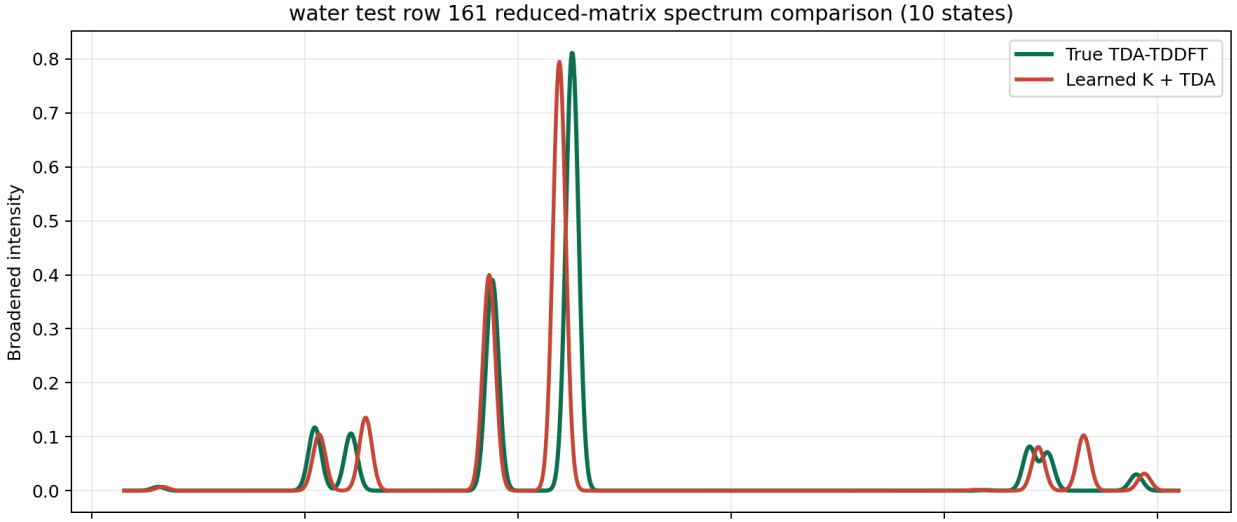
Water in Full Excitation Space - training set



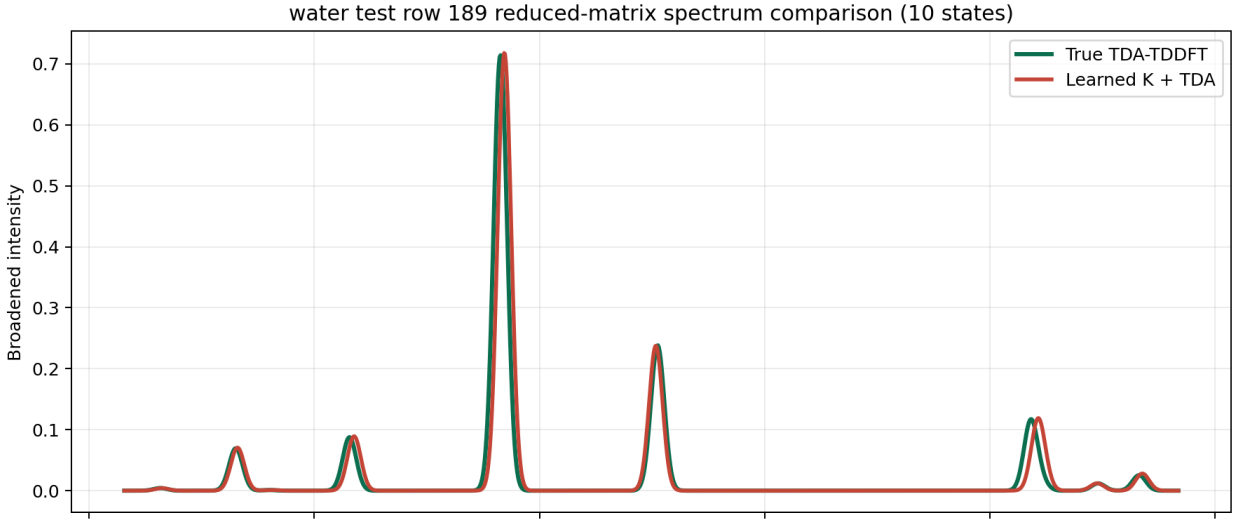
Speed up



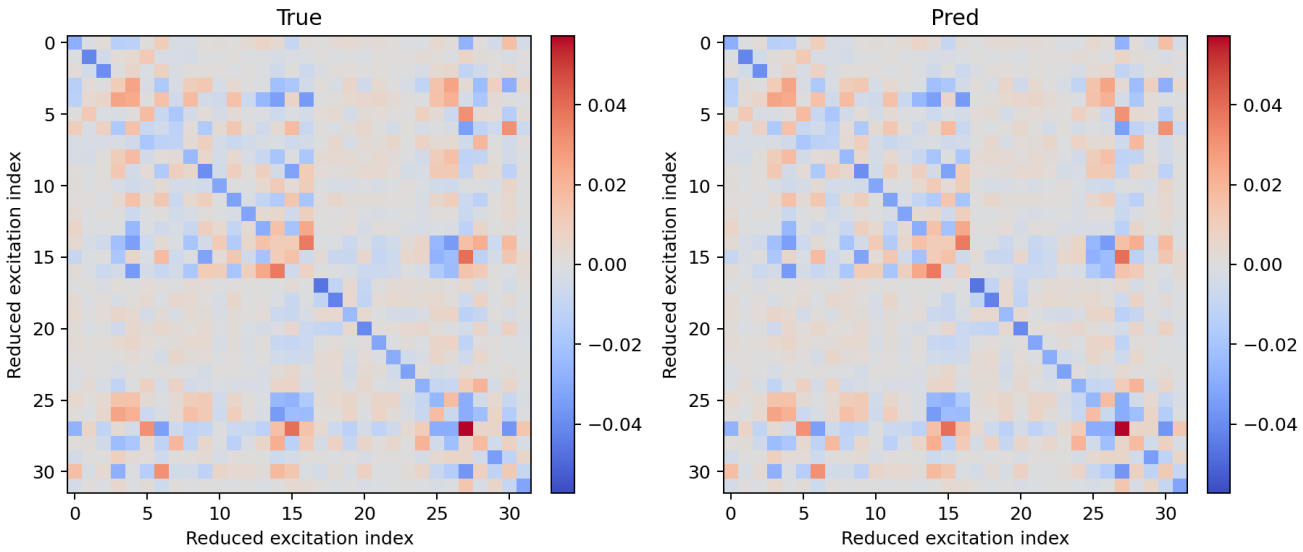
Water in Full Excitation Space - test set



Water in Full Excitation Space - test set



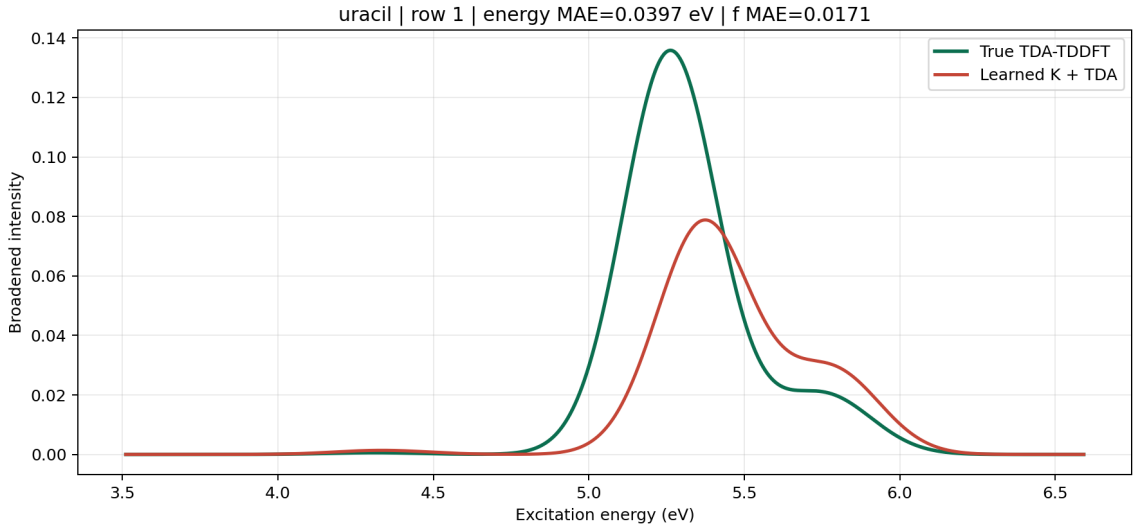
■ DL-TDDFT – A Test for Accelerating TDDFT



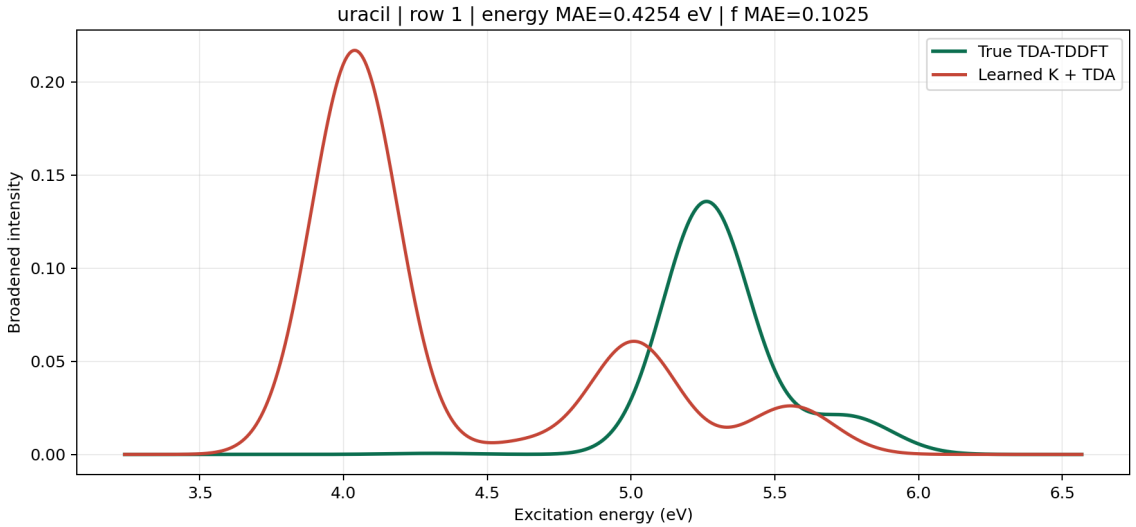
- **Good Enough? Right Enough? Differential Molecular Systems?**
- **Compared with SpaiNN?**
- **Phase Random?**

- **DF/Cholesky bipartite GNN**
- **Four-index AO/DF entry features**

Uracil in Excitation Subspace(32×32)



Uracil in Excitation Subspace(64×64)



■ Background – Inspiration From Claude Code

Water Molecule: Setup and Calculations

Computational Setup:

- **Molecule:** H_2O (water molecule)
- **Method:** TDDFT with various functionals
- **Basis Set:** 6-31G(d) and 6-31+G(d,p)
- **Functionals Tested:**
  - PBE (GGA functional)
  - B3LYP (Hybrid functional)
  - CAM-B3LYP (Long-range corrected hybrid)

Vibe Coding Approach:

- Used natural language to describe requirements
- AI assistant generated Python code for TDDFT calculations
- Automated spectrum plotting and analysis
- Interactive refinement of visualization

Comparison of Different Functionals

Key Observations:

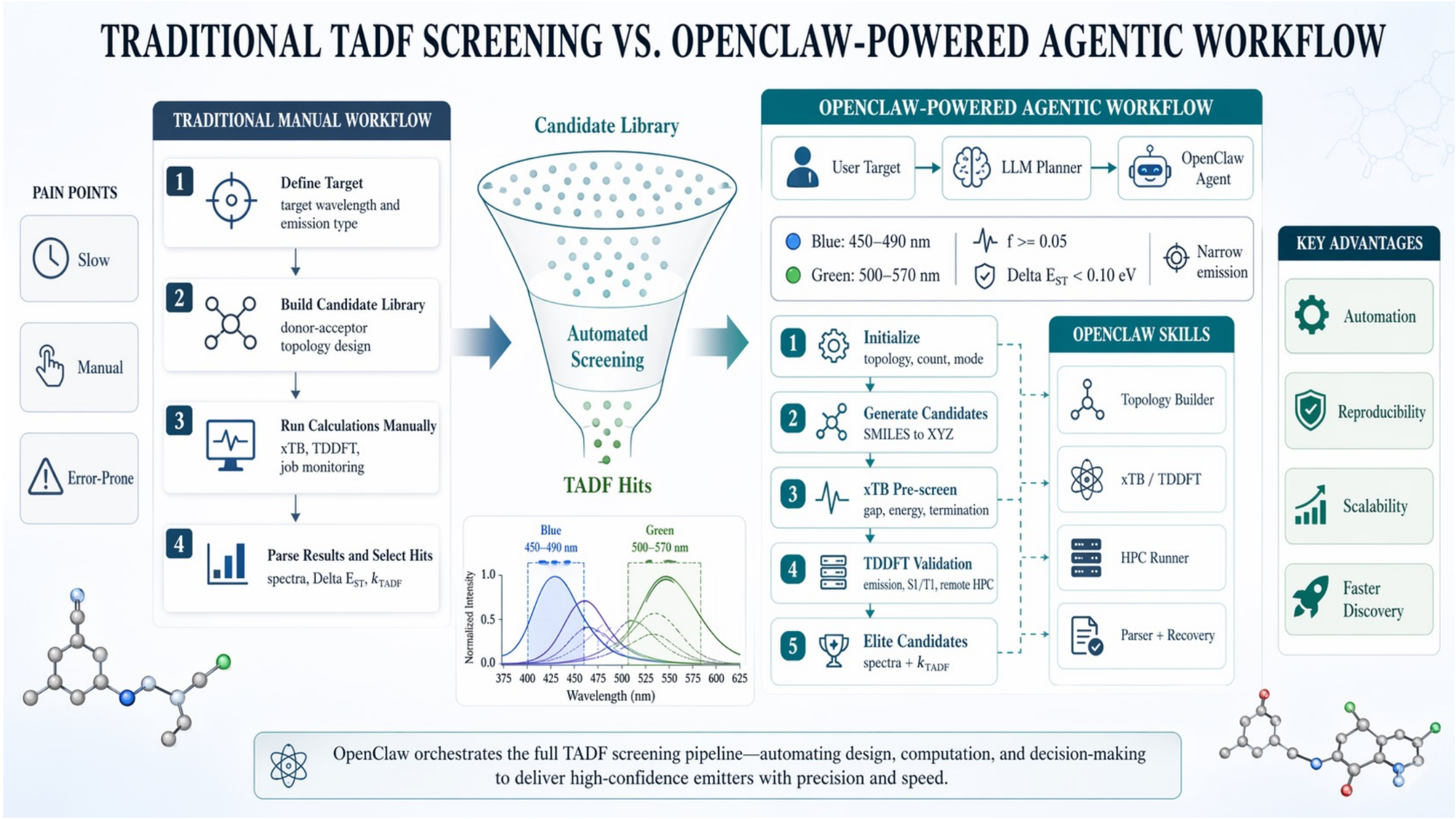
- PBE underestimates excitation energies
- B3LYP provides better agreement with experiment
- CAM-B3LYP improves for charge-transfer states
- Basis set effects are significant

Functional	Peak (eV)	Intensity
PBE	7.2	0.85
B3LYP	7.8	0.92
CAM-B3LYP	8.1	0.88
Exp.	7.9	-

Table 1: Comparison of excitation energies

■ Background – High Throughout Materials Design

github.com/silico-quantum/tadf-screening



■ Openclaw and Hermes – Basic Architecture

```
# Silico 🔮 - AI Research Partner

## Architecture

| Layer | Components |
|-------|------------|
| **Identity** | `SOUL.md` · `IDENTITY.md` · `USER.md` |
| **Memory** | `MEMORY.md` · `memory/*.md` |
| **Workflow** | `HEARTBEAT.md` · `AGENTS.md` · task plans |
| **Skills** | OpenClaw tools · chemistry workflows · automation modules |

## Runtime

- **Interface**: Feishu DM
- **Execution**: Local macOS environment + optional remote HPC
- **Models**: Configurable LLM backend
- **Automation**: Scheduled research and workflow jobs

## Key Capabilities

- Quantum chemistry workflows (DFT, TDDFT, xTB screening)
- Feishu docs / drive / wiki / messaging automation
- Web research, code execution, and repo maintenance
- Long-running task orchestration and progress tracking

## Self-Evolution (OpenSpace)

Every task teaches · Every failure improves
```

```
# Hermes △ – AI Computational Chemistry Partner

## Architecture

| Layer | Components |
|-------|------------|
| **Identity** | Hermes Agent · USER PROFILE · MEMORY |
| **Memory** | persistent memory · session history · reusable skills |
| **Tasks** | todo · cron jobs · delegated subagents |
| **Skills** | PySCF · xTB · Browser · Feishu · GitHub · visualization |

## Runtime

- **Host**: Local Hermes environment + Feishu DM
- **Local Compute**: PySCF, Python, matplotlib, file/workspace automation
- **External Access**: web search, browser automation, messaging tools
- **Model**: OpenAI GPT-5.4 (current session)

## Key Capabilities

- Computational chemistry workflows (PySCF, xTB, spectra, PES, MD analysis)
- Scientific scripting, plotting, and reproducible project pipelines
- Web lookup, browser verification, and literature cross-checking
- Sub-agent orchestration, file management, and result delivery

## Working Style

- Direct execution first, then concise reporting
- Prefer reproducible scripts over manual operations
- Scientific correctness over cosmetic imitation
- Fast iteration with explicit documentation of approximations

## Self-Evolution

Every task teaches · Every correction sharpens the workflow
```

https://github.com/HKUDS/OpenSpace

■ Openclaw – A Standard Skills Group For Quantum Chemistry Calculation

🧬 Core Skills

1. ⚛️ Structure & Sampling

- Molecular Sampler: Extract and sample molecular structures from cluster/ONIOM files. Union-Find molecule identification + distance-sorted neighbor sampling.
- RDKit Chemistry: 3D conformer generation (ETKDG), molecular descriptors (LogP, TPSA), and Gasteiger charge analysis.
- xTB Cluster MD: Semi-empirical MD (GFN-FF/GFN2-xTB) for organic molecular clusters with automated trajectory animation.

2. 🧪 Electronic Structure Computing

- PySCF: Python-based quantum chemistry. Supports Ground state (HF, KS-DFT), Excited states (LR-TDDFT, TDA), and Post-HF methods.
- MOMAP: helper for molecular photophysics and charge transport. Predict radiative/non-radiative (IC/ISC) rate constants and quantum yields.

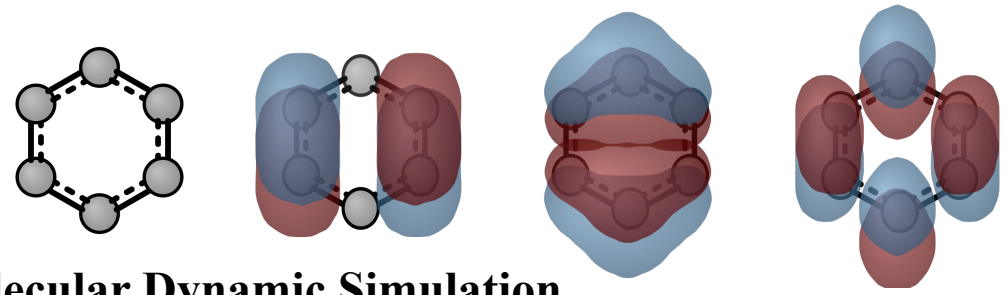
3. 📊 Analysis & Visualization

- Multiwfn: Advanced wave function analysis. Population analysis (Hirshfeld, ADCH, CM5), bond orders, and spectroscopy (UV-Vis, IR, Raman).
- xyzrender: Command-line driven, publication-quality molecular graphics. Supports transparent backgrounds, bond orders, and orbital rendering.
- Orbital Analysis: Streamlined workflow for MO composition, energy level diagrams, and isosurface generation.

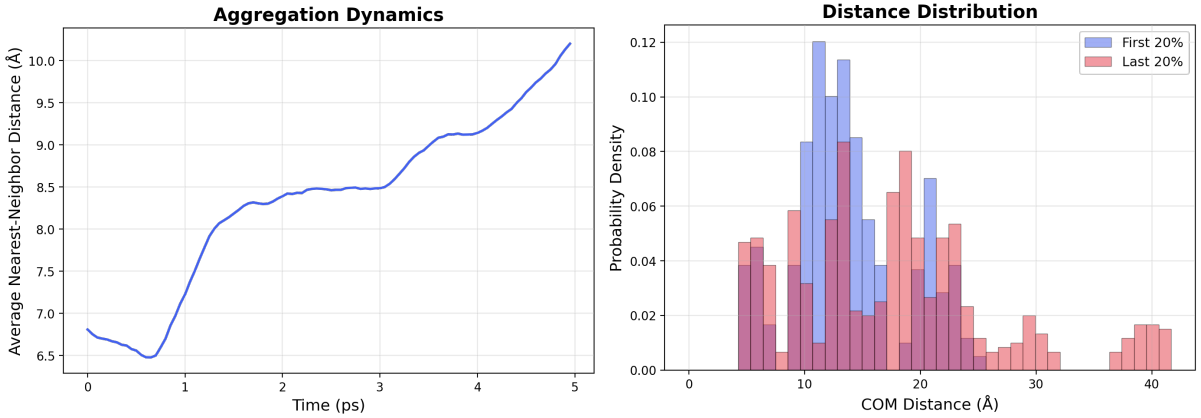
```
pyscf/
├── SKILL.md
├── VERSION_UPDATE.md
├── references/
│   ├── theory/
│   │   ├── pyscf-advanced.md
│   │   ├── pyscf-api-reference.md
│   │   └── pyscf-jax-integration.md
│   └── practice/
│       ├── 2d-potential-energy-surface.md
│       ├── emission-spectrum-guide.md
│       └── emission-spectrum-workflow.md
├── scripts/
│   └── dft_calculation.py
└── tools/
    ├── analysis.py
    ├── cascf.py
    ├── ccsd.py
    ├── dft.py
    ├── geometry.py
    ├── mp2.py
    ├── pes.py
    ├── scf.py
    ├── spectrum.py
    └── tddft.py
```

■ Openclaw and Hermes – A Standard Skills Group For Quantum Chemistry Calculation

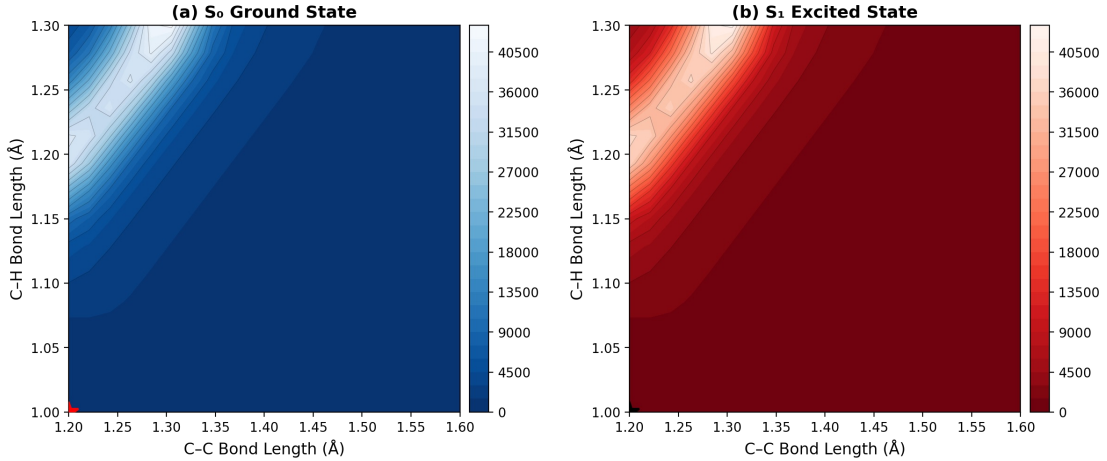
Basic Ground State Calculation



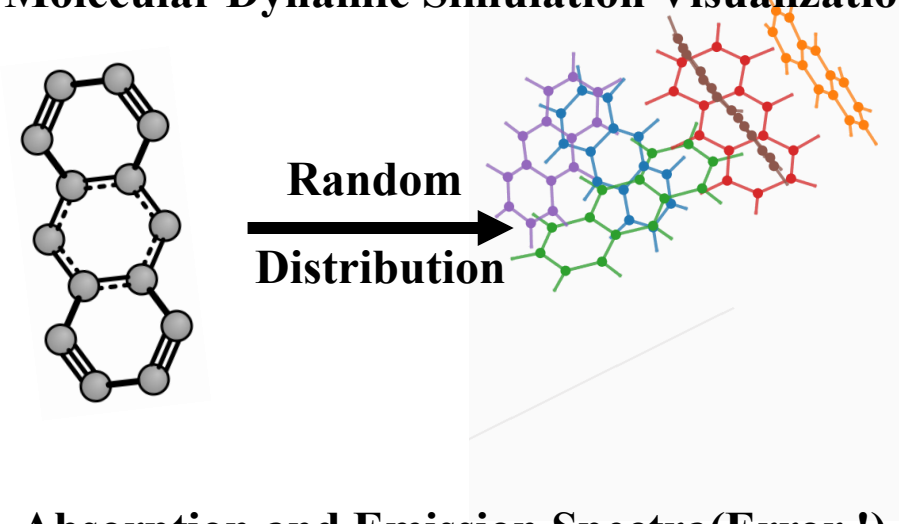
Molecular Dynamic Simulation



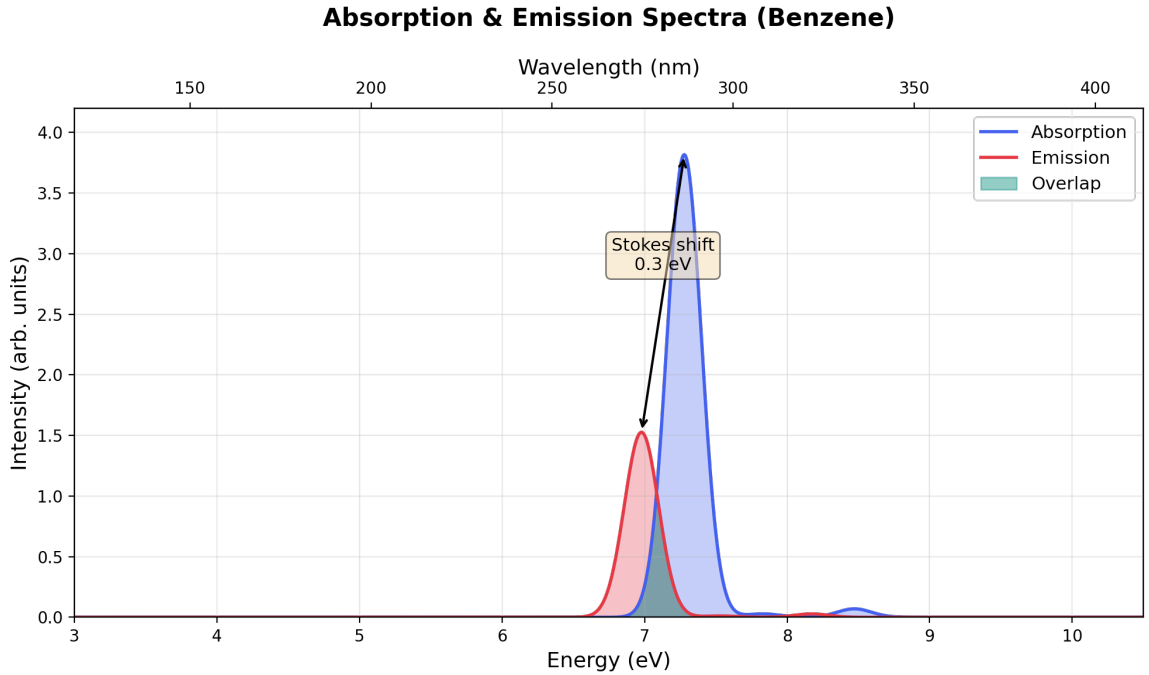
Potential Energy Surface



Molecular Dynamic Simulation Visualization

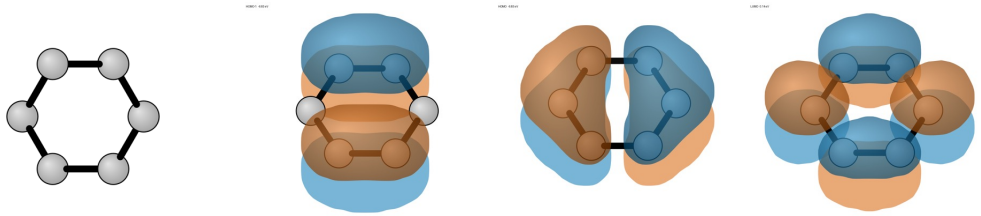


Absorption and Emission Spectra(Error !)

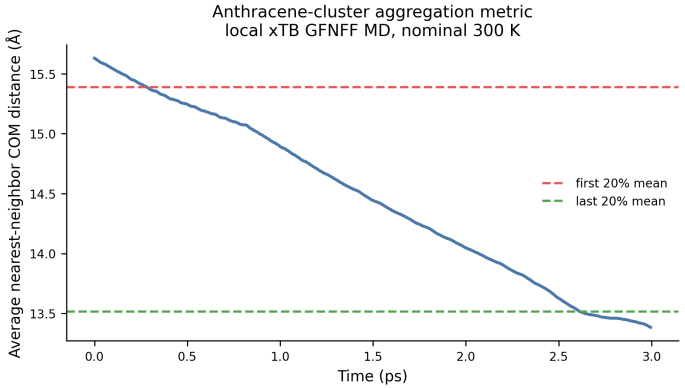


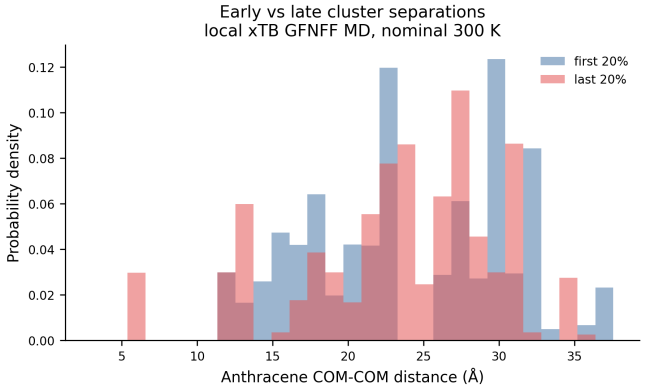
■ Openclaw and Hermes – A Standard Skills Group For Quantum Chemistry Calculation

Basic Ground State Calculation

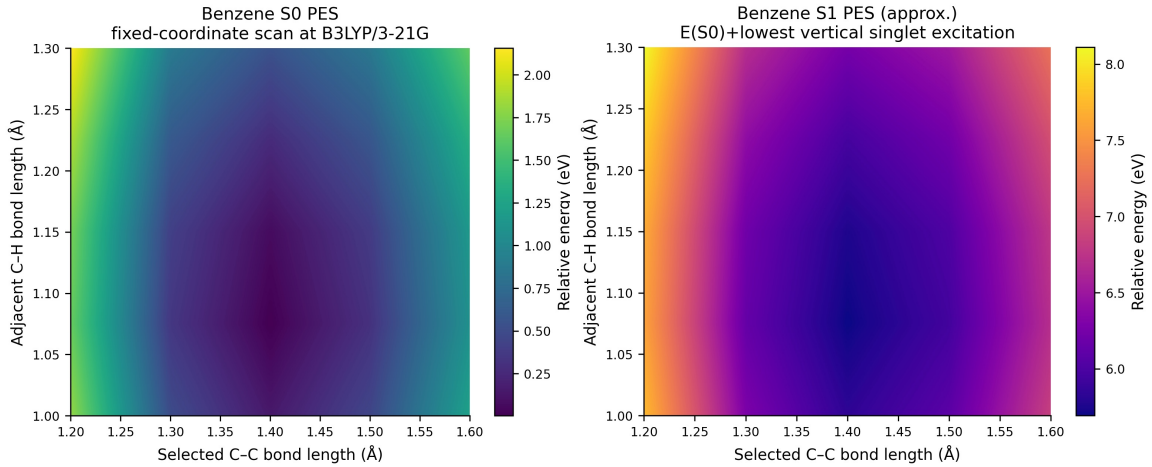


Molecular Dynamic Simulation

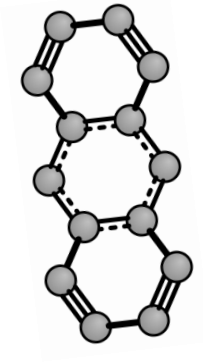




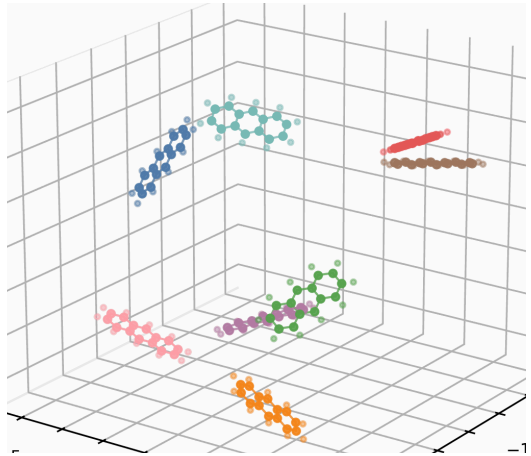
Potential Energy Surface



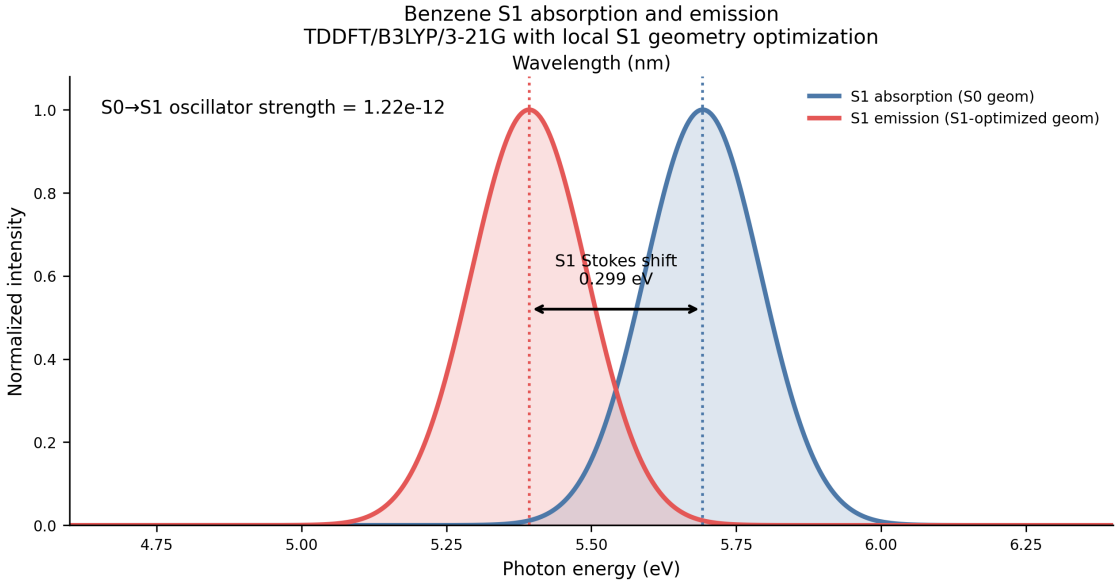
Molecular Dynamic Simulation Visualization



Random
Distribution



Absorption and Emission Spectra



■ Openclaw and Hermes – Skill For TADF Materials Design

```
Stage 0: Initialization
  └─ screening_workflow_initializer.py
     └─ Confirm: topology, sample count (default 10k), interaction mode

Stage 1: Candidate Generation
  ├─ build_da_topology_library.py  (D-A, D-A-D, A-D-A, D-pi-A...)
  └─ SMILES → .xyz structure generation (export to manifest.csv)

Stage 2: xTB Pre-screening
  ├─ Input: manifest.csv (idx, name, xyz_path)
  └─ run_xtb_batch_manifest.py → xtb_progress.csv
      Required columns: idx, name, xyz_path, status, detail,
                        total_energy_eh, homo_lumo_gap_ev, normal_termination
  ⚠ If xTB ok=0 → STOP and diagnose

Stage 2B: TDDFT-xTB Wavelength Filter (optional)
  └─ run_tddft_xtb_filter.py
      Filter: emission in target range, f ≥ threshold

Stage 3: TDDFT Validation (remote)
  ├─ S0 optimization
  ├─ Excited-state optimization (S1 for TADF/Fluorescence; T1 for Phosphorescence)
  ├─ Vertical emission
  └─ Gate: ΔE_ST < 0.10 eV, f ≥ 0.05, k_TADF > 10⁴ s⁻¹

Stage 4: Elite Candidates → downstream analysis
```


```
{
  "project": {
    "name": "green-tadf-screening-10k",
    "compute_tier": "local_basic",
    "interaction_mode": "interactive_chat",
    "question_mode": "blocking",
    "max_question_rounds": 3
  },
  "system_hardware_profile": {
    "hardware": {
      "platform": "macOS-26.1-arm64-arm-64bit",
      "os": "Darwin",
      "cpu_cores_logical": 10,
      "gpu": { "available": false, "count": 0, "names": [] }
    },
    "hpc_slurm": {
      "remote_hpc_hint": null,
      "slurm_detected": false,
      "sbatch_in_path": false,
      "squeue_in_path": false,
      "slurm_env_present": false
    }
  },
  "photophysical_targets": {
    "emission_range_nm": [500.0, 570.0],
    "emission_type": "TADF",
    "spectrum_width_requirement": "Narrow/High Color Purity",
    "default_fwhm_ev": 0.15
  },
  "fragment_database": {
    "builtin_sources": ["DeepChem", "PubChem"],
    "augment_with_custom_db": null,
    "custom_db_paths": []
  },
  "assembly_topology_rules": {
    "topology_preferences": ["D-A", "D-A-D"],
```

■ Openclaw and Hermes – An Example For Bule Light OLED/TADF Materials

/skill tadf-screening

11:20

已切到 `tadf-screening` 工作流。
先把入口条件定掉，我就能直接往下跑：

1. 目标发光类型, TADF / fluorescence / phosphorescence
2. 目标发射范围, 例如 450–490 nm
3. 是否要求窄带, 以及你接受的带宽标准
4. 拓扑, `D-A` / `D-A-D` / `A-D-A` / `D-pi-A` / `D_n-A` / 混合
5. 初始样本数, 默认 10000
6. 远程资源用哪个 profile, 比如 marcus；如果缺 `xtb4stda/stda`，我是否可以尝试安装
7. 监控节奏, 默认每 5 分钟检查、每 10 分钟汇报

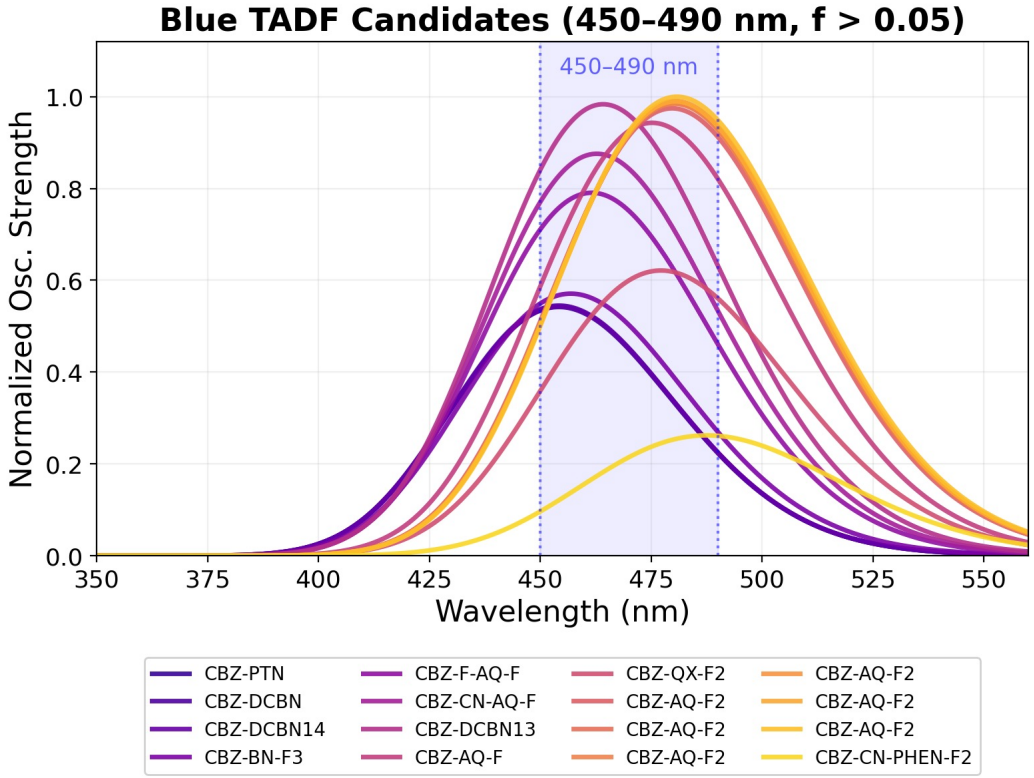
如果你想省事，直接回我一句：
“窄带蓝光 TADF，450-490 nm，D-A-D，10000，marcus，允许安装缺失工具，默认监控”
我就开干。

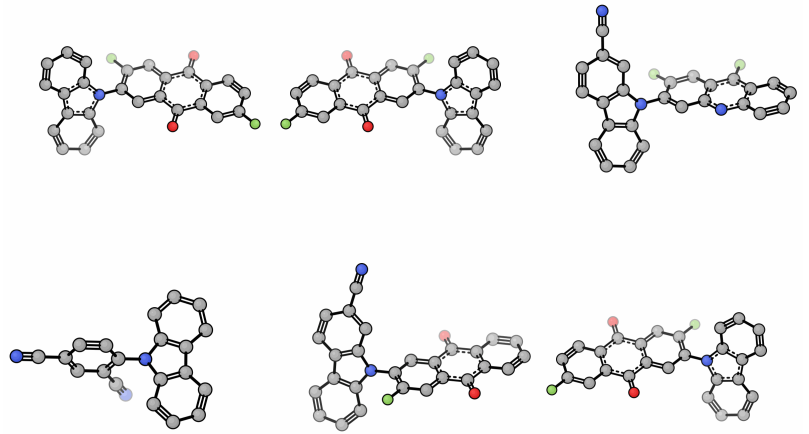
窄带蓝光 TADF，450-490 nm，D-A-D，10000，marcus，允许安装缺失工具，默认监控，注意marcus的进入方式是ssh ->marcus2再ssh marcus，使用slurm进行远程的所有计算

好，我已经按这个参数把入口配好了：

- profile: `yuanjiao_marcus`
- 路径: `ssh marcus2 -> ssh marcus -> Slurm`
- 监控: 已设为每 5 分钟检查、每 10 分钟汇报
- 初始化配置已生成: `skills/tadf-screening/workflow_config_blue_10k_20260422.json`
- `BATCH_STATUS.md` 也已切到这次 batch

现在只差一个硬门槛才能正式往下跑：
Stage 2 的 empirical Stokes shift 取多少 eV?
我建议先用 **fixed 0.5 eV**，这和我们前面蓝光筛选流程一致，也最稳。
你直接回我一句 **“用 0.5 eV”**，我就继续进下一步。
补一句，初始化器在本地只探测到 PySCF，但这不影响，**Stage 3 仍会按你的要求走 marcus 上的 Gaussian + Slurm。**





■ Openclaw and Hermes – Cron, Heartbeat, Memory, Dreaming and Self-evolution

- Openclaw-planer and supervisor

Cron

- Long-term tasks
- Accurate time point
- Isolated mode(work in sandbox)

Heartbeat

- Short-term tasks
- Ambiguous time point
- Only work in main process

Memory

Human
→ daily Memory
→ long-term memory
→ semantic retrieval → answer

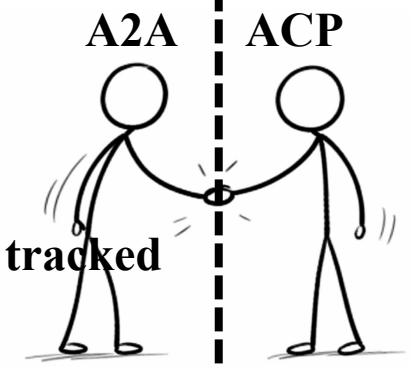
Dreaming

past memory files
→ semantic recall happens
→ recalled chunks are logged
→ frequently used / high-signal chunks get tracked
→ future recall can become more targeted

- Hermes-executor and worker

Self-evolution

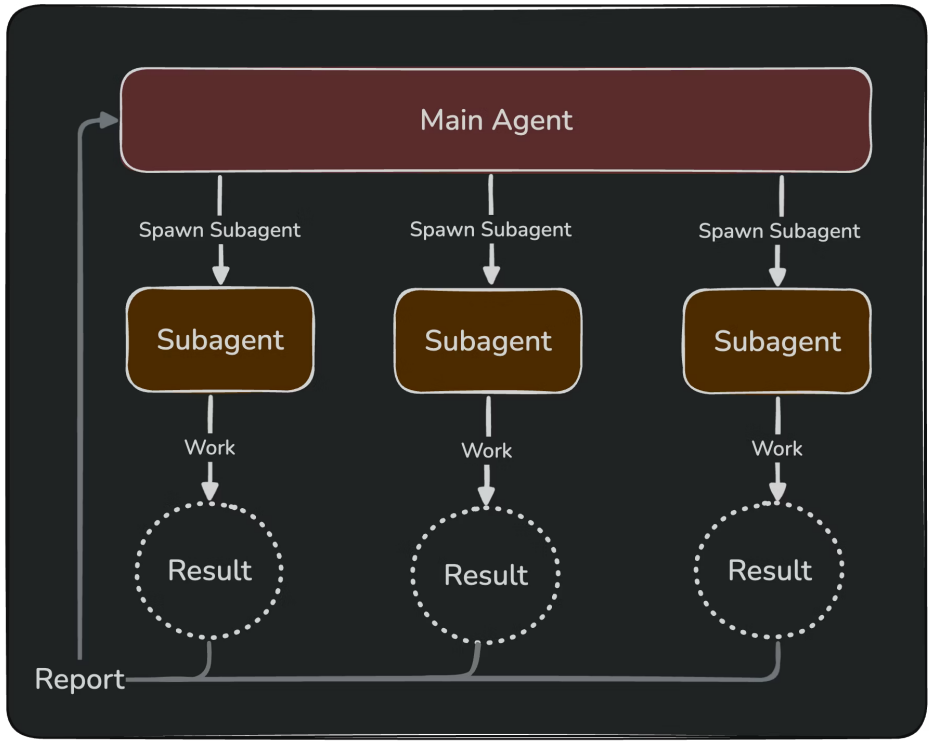
Real execution
→ feedback from success and failure
→ distillation of reusable lessons
→ memory for facts, skills for methods, workflows for reproduction
→ correction and compression of weak or stale structure
→ better future execution



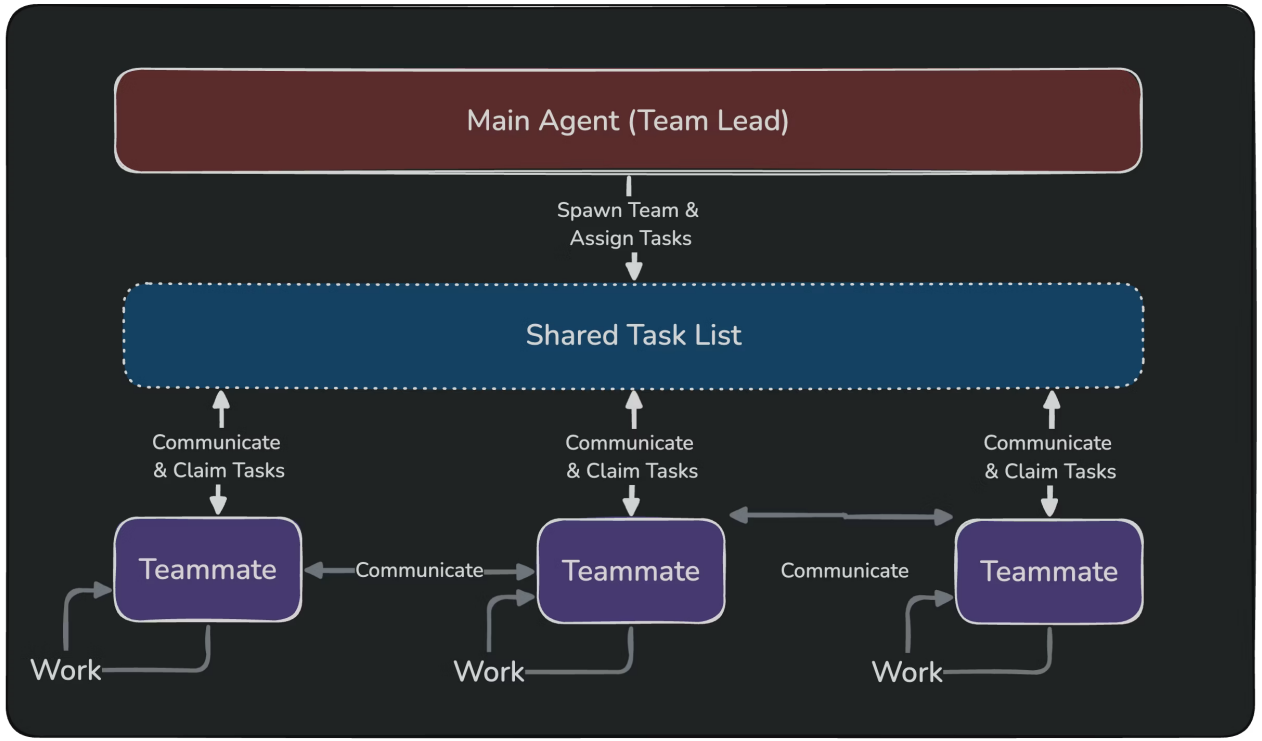
Under Exploring…

■ Openclaw and Hermes – Suitable Hardness, Expert Agents

Subagent



Agent Teams



https://code.claude.com/docs/en/agent-teams

- penggroup.agent with Agent Teams!
- Lammps AI !—More Suitable Harness, Better Agent!
- Expert agents for ab initio process !

Github.com/karpathy/autoresearch

Github.com/datawhalechina/hello-agents