Why I did this? What I learned?

Recently, I've got interested in Physics-Informed AI models . After the recent exit from LG Electronics (I resigned in mid-October, 2025) , I've been diving deep into the world of Physics-Informed NN models, analyzing the underlying theories, and following up models such as Hamiltonian NN (HNN, brought by Sam Greydanus et al., arXiv 2019), Diffusive HNN (D-HNN by Andrew Sosanya et al., arXiv 2022), discrete autograd (DGNet by Takashi Matsubara et al., NeurIPS 2020), and Neural Symplectic Form (by Yuhan Chen et al., NeurIPS 2021) with analyzing their theories and evaluating their github codes.

After thorough testing altogether for HNN, D-HNN, DGNet, their theories and codes are well prepared, the theories behind are robust and sound, convincing me those actually work! Here is one example for comparison between HNN, D-HNN, and DGNet which were trained/evaluated for the same "experiment-real" dataset and on the same torch context. See Figure 1.

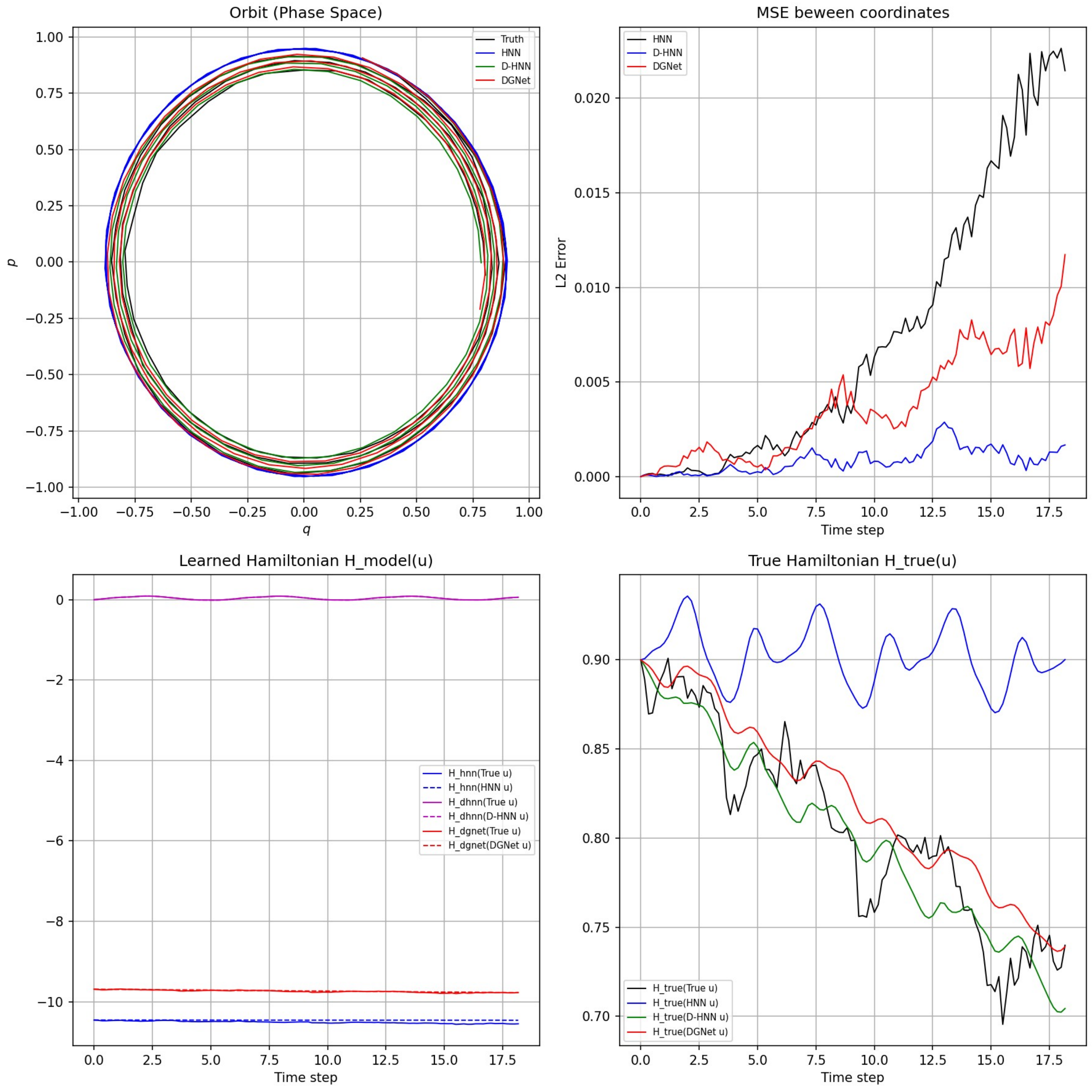


Figure 1. Comparison of evaluation across HNN, D-HNN, and DGNet for the 'experiment-real' problem.

Next then, I moved to analyze the theories formulating the Neural Symplectic Form. I found the theory itself is mathematically and physically sound, correct, and robust and thought this should work! After this, I started analyzing, training, and evaluating their github code. One great thing Y. Chen delivered is that she included, in her github repo, all valuable models such as Hamiltonian NN (HNN), Lagrangian NN (LNN), Neural Symplectic Form (SYM), Skew Matrix Learning (SKEW), and Neural ODE (NODE). All implementations of the models each are sound and have no errors.

I trained all the models in the repo using the provided train codes each, then started to evaluate. Then, I was casted into an endless agonizing pain. The evaluated result for HNN was really weird in the context of my previous experiences with HNN, D-HNN, and DGNet. The HNN result was hard to believe as it is. Furthermore, more sophisticated numerical device like LNN fails to predict any and seemingly showed the worst performance. I couldn't believe this at all, especially for LNN, one of the most sophisticated designs of NN! Here are orbit results compared for the mass-spring. See Figure 2. Though I've found that the initial state used for NODE orbit prediction is NOT the same as the ground truth orbit, but this minor mistake does not annoy me. What frustrated me was that HNN fails to follow the ground truth (e.g. v2 in my evaluation), and, more seriously LNN showed approximately only half the ground truth orbit in both the reported and my evalutions in frequency behavior. Please pay attention to the time window for HNN and LNN [0, 8] in the paper but they used [0, 5] for the other models and ground truth and I still kept [0, 5] for all models. Besides, the magnitudes of the reported HNN and LNN orbits each does not coincide with my evaluation. They mentioned that they used double precision in the presented data, where the github code is single-precision. Even so, I think these weird behavior cannot be explained with the numerical precision they mentioned. See they also plotted with [0, 8] time interval which seems to exhibit a pattern similar to the ground truth orbit in [0, 5] time window!

The situation becomes much worse for the chaotic system like the double-pendulum system as shown in Figure 3.

So, I thought, thought, and thought of what's wrong or where wrong for HNN and LNN. And reached the following two points.

1. First Point

A question hit me:
"Is the data normalization OK without breaking the laws of physics?"
I tried to find a logical answer to this.

1) The Physics: Symplectic Structure

The core of Hamiltonian mechanics lies in the (q, p) phase space, which possesses a 'symplectic structure'. This is represented by the 2-form,

$\omega = dq \wedge dp$.

The property of this structure is that it is 'closed'. Its exterior derivative is zero: $d\omega=0$. This is where energy conservation stems from and gives rise to Hamilton's equation: $dq/dt = \partial H/\partial p$, $dp/dt = - \partial H/\partial q$.

HNNs are trained to learn this $d\omega=0$ constraint. This is where energy conservation stems from.

2) Normalization: It Preserves Structure (When Done Right!)

A standard normalization used (like max_abs scaling) is a linear transformation $z'=Mz$, where M is a constant diagonal matrix (fixed scaling factors across the entire dataset).

The transformed symplectic form would be $\omega' = dq' \wedge dp'$. Since M is a constant:

$\omega' = dq' \wedge dp' = (M_q dq) \wedge (M_p dp) = (M_q M_p)\, \omega$

To verify if ω' is closed, the exterior derivative d is taken:

$d\omega' = d((M_q M_p)\omega)$

The constants M_q, M_p commute with the operator d:

$d\omega' = (M_q M_p) d\omega$

Since the original $d\omega=0$,

$d\omega' = 0$

3) The Real Problem: Variable Transformations (e.g., Double Pendulum)

What if the transformation matrix M depends on the coordinates q or p ?
(1) Mass-Spring System: $p = M v$, where the mass matrix $M = \mathrm{diag}(m_1, m_2)$ is a constant. (Safe!)
(2) Double Pendulum: $p = M v$. The mass matrix $M=M(q)$ depends on the angle q !

Therefore, for the mass-spring problem, (q, v) space can be treated as sort of a scaled space for (q, p). Training HNN in $z=(q, v)$ space will embed the symplectic structure by forcing some Hamiltonian-like equation of motion: $dq/dt = \partial H_{NN}/\partial v$, $dv/dt = - \partial H_{NN}/\partial q$, where H_{NN} is a scalar outputted by HNN and HNN will learned Hamiltonian function in the scaled space (q, v). When scaled back to the original (q, p) space, then the (q, p) orbit equation will satisfy the Hamilton's equations, $dq/dt = \partial H/\partial p$ and $dp/dt = - \partial H/\partial q$ where H will corresponds to the true Hamiltonian that would have been learned if trained on (q, p) space, because $p = Mv \propto v$. And the data normalization

x_input_scaled = B^{-1} x_input,
y_target_scaled = A^{-1} y_target,
$A = B$ = diag(max_abs(y_target_v_1, y_target_v_2, y_target_a_1, y_target_a_2)),
($v = dq/dt$ = Velocity (Linear/Angular), $a = dv/dt$ = Acceleration (Linear/Angular)

used in the Y. Chen's dataset preparation and train codes and their paper as well, will not break the physics for the mass-spring system.

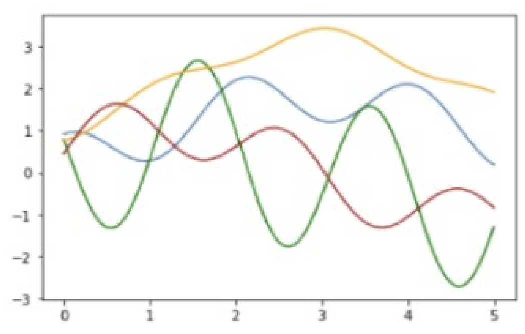
What if working on the double pendulum problem with $z=(q, v)$ space?
In this case, however, $v=M(q)^{-1}p$ would give $d\omega(q, v)\neq 0$ naturally! Forcing a symplectic structure (i.e., forcing $d\omega(z)=0$) to HNN(z) would give rise to $dq/dt = \partial H_{NN}/\partial v$, $dv/dt = - \partial H_{NN}/\partial q$ *and* HNN would still output some constant scalar for the input (q, v). But this time, we have

$dp/dt = d(Mv)/dt = (dM(q)/dt)v + M\, dv/dt = (dM(q)/dt)v + Ma$,

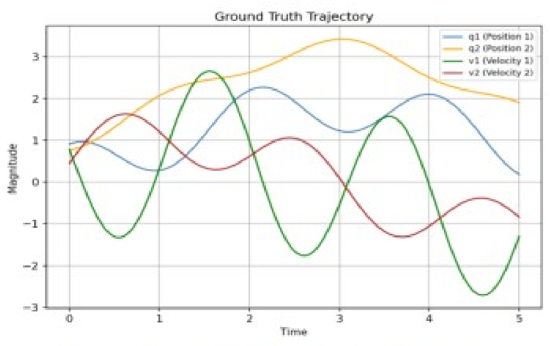
which means we cannot simply scale (q, v) back to (q, p) and the learned Hamiltonian-like function in (q, v) will not guarantee Hamilton's equation in (q, p). The above data normalization would be no rescue for this. The data will, anyway, be fitted like the brute force NN ODE solver NODE. But the generalization is an open question.

For the double-pendulum system, I think we should train HNN directly in (q, p) space, which will not break physics.

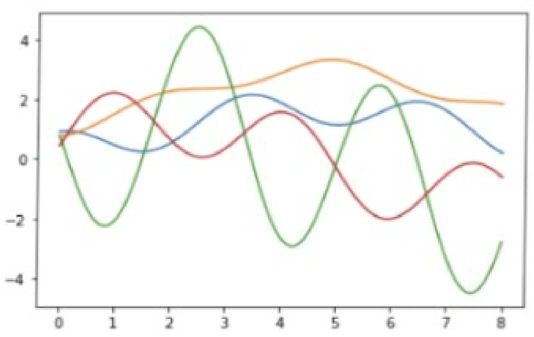
Mass-Spring Problem



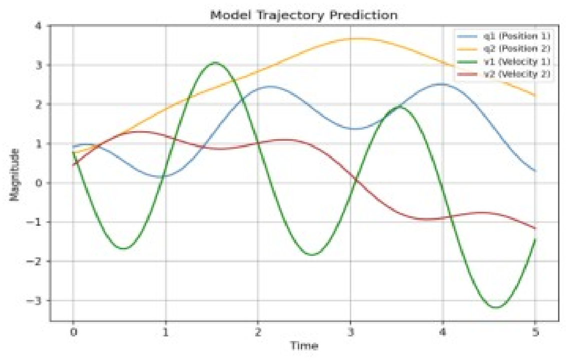
(a) Ground truth



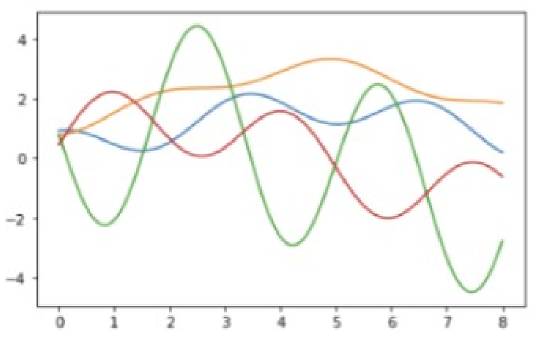
Ground Truth Orbit using the initial state set in Y. Chen's github code



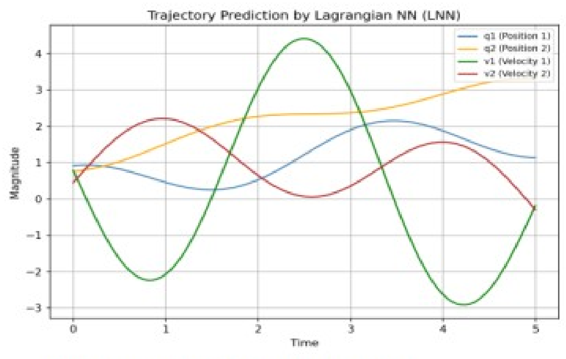
(c) HNN



Orbit prediction using HNN trained ‘as-is’ by Y. Chen’s github code



(d) LNN

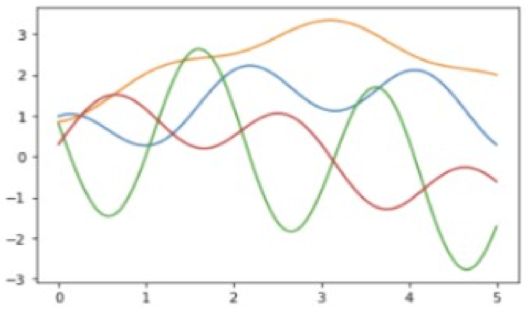


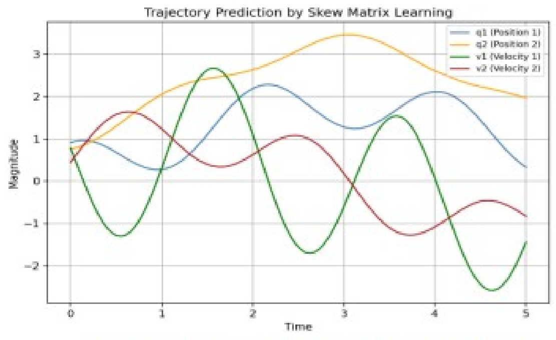
Orbit prediction using LNN trained ‘as-is’ by Y. Chen’s github code

From Y. Chen et al.’s NeurIPS (2021) paper

Figure 2(a). Comparison of ground truth orbit, HNN, and LNN between the reported and the actual evaluation for the identical initial state vector (q1, q2, v1, v2) in (q, v)-space.

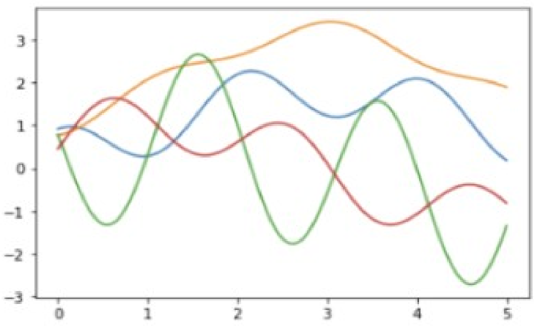
Mass-Spring Problem

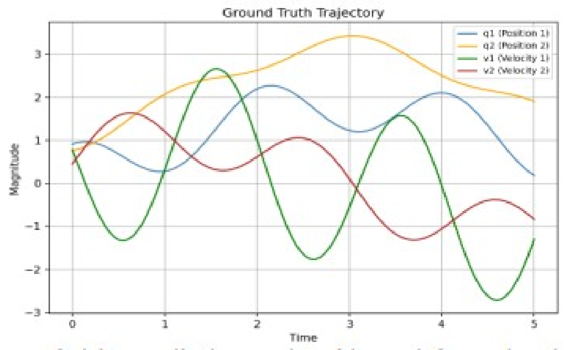




(e) Skew Matrix Learning

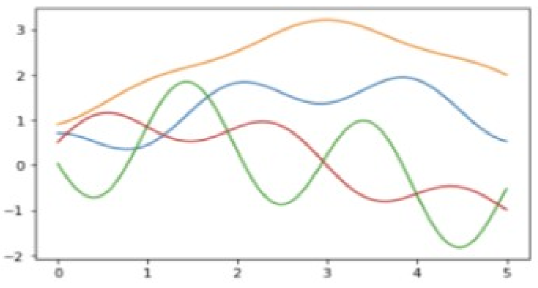
Orbit prediction using Skew Matrix Learning (SKEW) using Y. Chen's github code

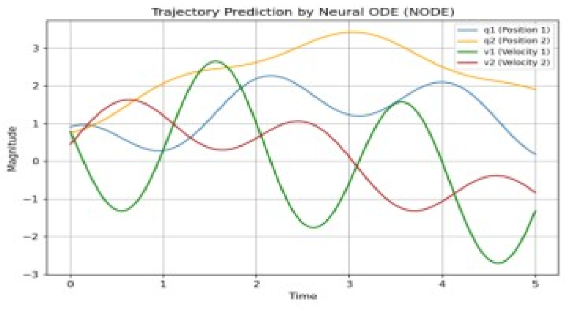




(f) Neural Symplectic Form

Orbit prediction using Neural Symplectic Form (SYM) trained 'as-is' by Y. Chen's github code





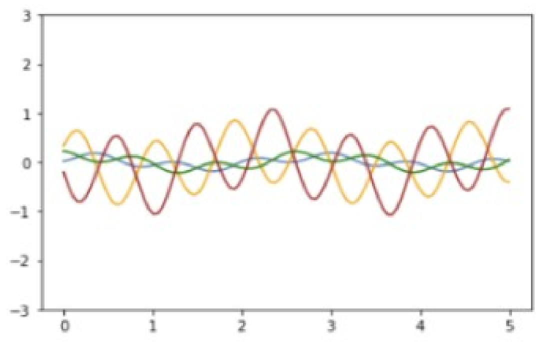
(b) NODE

Orbit prediction using NODE trained 'as-is' by Y. Chen's github code

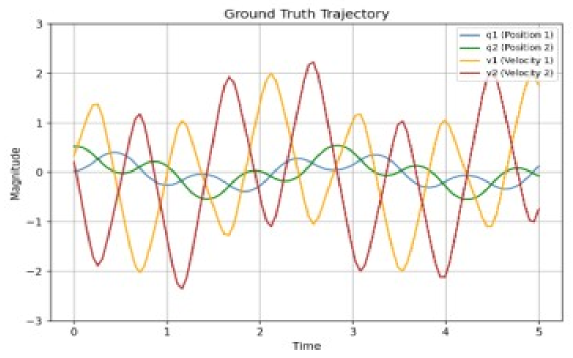
From Y. Chen et al.'s NeurIPS (2021) paper

Figure 2(b). Comparison continued fo Skew Matrix Learning (SKEW), Neural Symplectic Form (SYM), and Neural ODE (NODE) between the reported and the actual evaluation for the identical initial state vector (q1, q2, v1, v2) except for NODE in the paper.

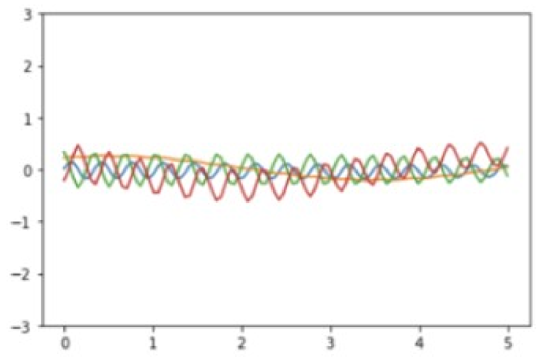
Double-Pendulum Problem



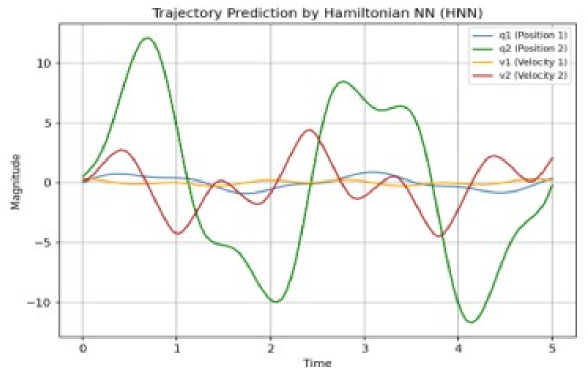
(a) Ground truth



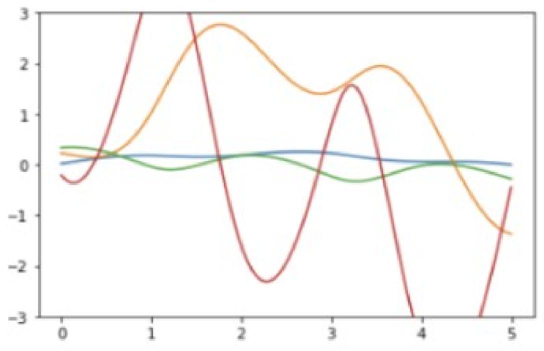
Ground Truth Orbit using the initial state set in Y. Chen's github code



(c) HNN

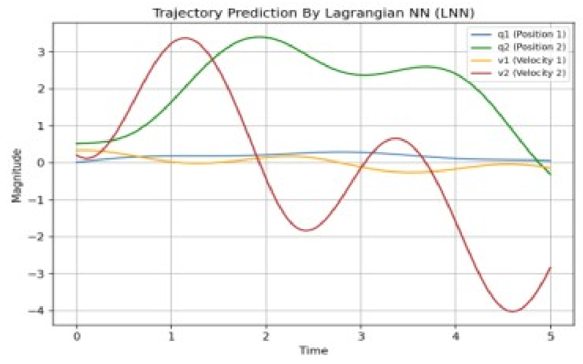


Orbit prediction using HNN
Trained 'as-is' by Y. Chen's github code



(d) LNN

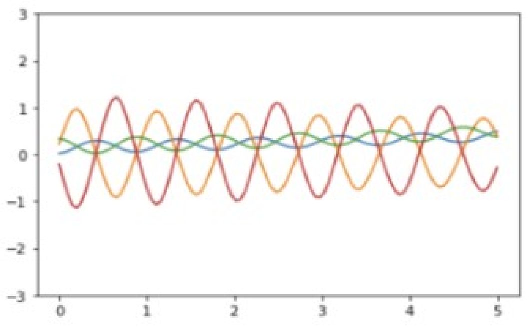
From Y. Chen et al.'s NeurIPS (2021) paper



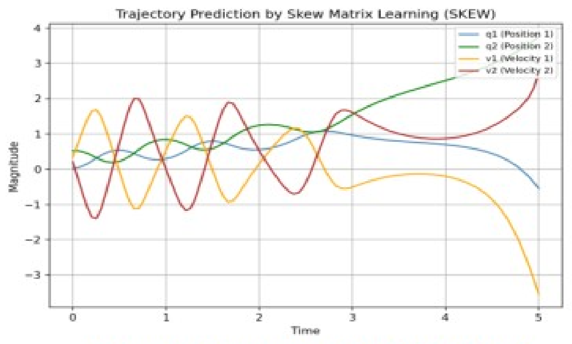
Orbit prediction using LNN
trained 'as-is' by Y. Chen's github code

Figure 3(a). Comparison of ground truth orbit, HNN, and LNN between the reported and the actual evaluation. Note that the initial states are different between the reported and my evalutions. Because I couldn't figure out the initial state vector used in the paper. I used the initial state set in the codes, instead. But, I think we should observe similar frequency patterns at least!

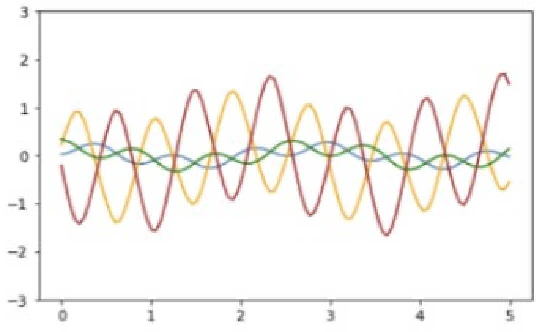
Double-Pendulum Problem



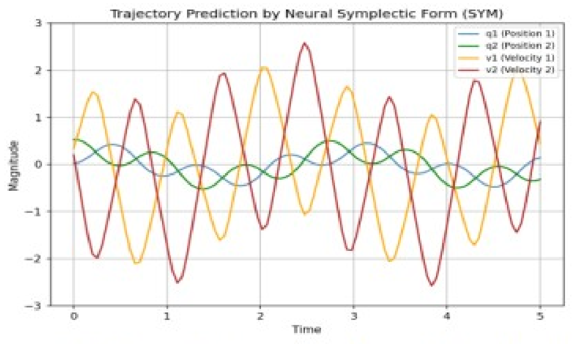
(e) Skew Matrix Learning



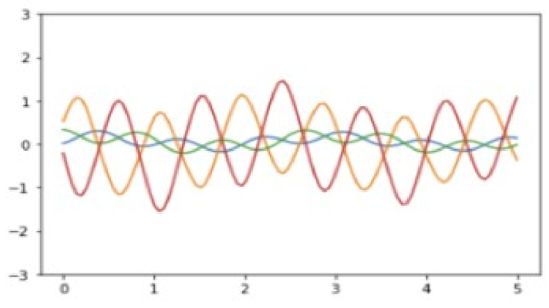
Orbit prediction using Skew Matrix Learning (SKEW) using Y. Chen's github code



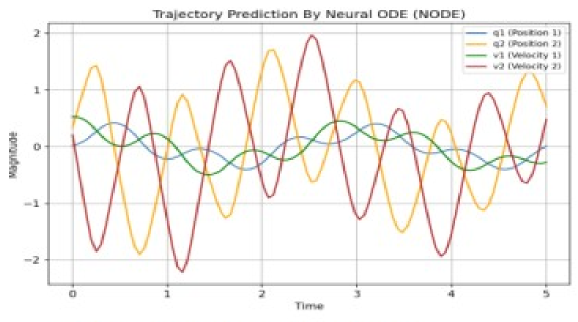
(f) Neural Symplectic Form



Orbit prediction using Neural Symplectic Form (SYM) trained 'as-is' by Y. Chen's github code



(b) NODE



Orbit prediction using NODE trained 'as-is' by Y. Chen's github code

From Y. Chen et al.'s NeurIPS (2021) paper

Figure 3(b). Comparison continued for Skew Matrix Learning (SKEW), Neural Symplectic Form (SYM), and Neural ODE (NODE). Because I couldn't figure out the initial state vector used in the paper. I used the initial state set in the codes, instead. But, we should observe similar frequency patterns at least!

2. Second Point

The first point might be a source of errors for HNN, but, of course, NOT for LNN which itself is mathematically and physics-wise formulated in (q, v) space. I wanted to tackle the HNN issue first. I did prepared (q1, q2, p1, p2) dataset from (q1, q2, v1, v2) dataset generated in Y. Chen's data preparation subroutine, adopting the same data normalization strategy (i.e., A=B=diag(max_abs(y_target_v_1, y_target_v_2, y_target_f_1, y_target_f_2) in which $f = dp/dt$ = Force (Linear(i.e, Force) or Angular(i.e., Torque). Then I re-trained HNN for the mass-spring and the double-pendulum system.

The result???
No progress at all, only the magnitude changed by using p instead of q and the weird pattern still persisted!
What the …
I felt I got trapped in the very, very frustrating situation.

A few days later, however, something popped into my head!

"What if scaling the input data and the output data separately? This will make the situation something different."

I executed the idea with the following independent scaling.

For HNN (q, p):
x_input_scaled = B^{-1} x_input(q_1, q_2, p_1, p_2),
y_target_scaled = A^{-1} y_target(v_1, v_2, f_1, f_2),
A = diag(max_abs(y_target_v_1, y_target_v_2, y_target_f_1, y_target_f_2)),
B = diag(max_abs(x_input_q_1, x_input_q_2, x_input_p_1, x_input_p_2)
($v = dq/dt$, $f = dp/dt$)

For the other models (q, v):
x_intput_scaled = B^{-1} x_input(q_1, q_2, v_1, v_2),
y_target_scaled = A^{-1} y_target(v_1, v_2, a_1, a_2),
A = diag(y_target_v_1, y_target_v_2, y_target_a_1, y_target_a_2),
B = diag(x_input_q_1, x_input_q_2, x_input_v_1, x_input_v_2)
(y_target_$v_{1,2}$ = x_input_$v_{1,2}$, $v = dq/dt$, $a = dv/dt$)

Actually, this scaling strategy is something of a consensus widely accepted in the data science community. I trained also HNN (q, v) using the above scheme, for comparison with HNN (q, p).

The Result? Hooray!!!
All problems disappeared and I was finally able to compare the true performances across the models.

The real aftermath:

Winners of orbit prediction performance:
- **Gold** Medalist: **Lagrangian NN (LNN)**
- **Silver** Medalist: **Neural Symplectic Form (SYM)**
- **Bronze** Medalist: **Neural ODE (NODE)**

But, the difference between these seems to be marginal.
One thing that surprised me is the good performance of NODE which is the simplest of them all. The Hamiltonian NN (HNN) seems to need improvement, especially to cope with the complex, chaotic system.

Neural Symplectic Form looks to be successful in coping with and overcoming this hurdle. It provides a way to learn Hamiltonian physics in terms of (q, v). Imagine that we have videos of recording some motion of objects in the real world. As demonstrated by Sam Greydanus, the HNN family can learn physics from images. From consecutive time frames taken with the calibrated cameras, position and velocity can be extracted but not momentum.

I think Y. Chen and her collaborators did a great job. However, I hope they correct the data reported in the paper and the data scaling in the code for future readers, so they do not repeatedly put themselves in agonizing pain like me.

I appreciate S. Greydanus, T. Matsubara, and Y. Chen for sharing their codes on GitHub.

Jae Hoon (Daniel) Lee, Ph.D.

P.S.: I also found that the angle coordinate configuration and the ODEs for the double-pendulum problem in Y. Chen’s paper are incorrect, whereas the GitHub code uses the correct ODEs.

<written on 2025-12-20>

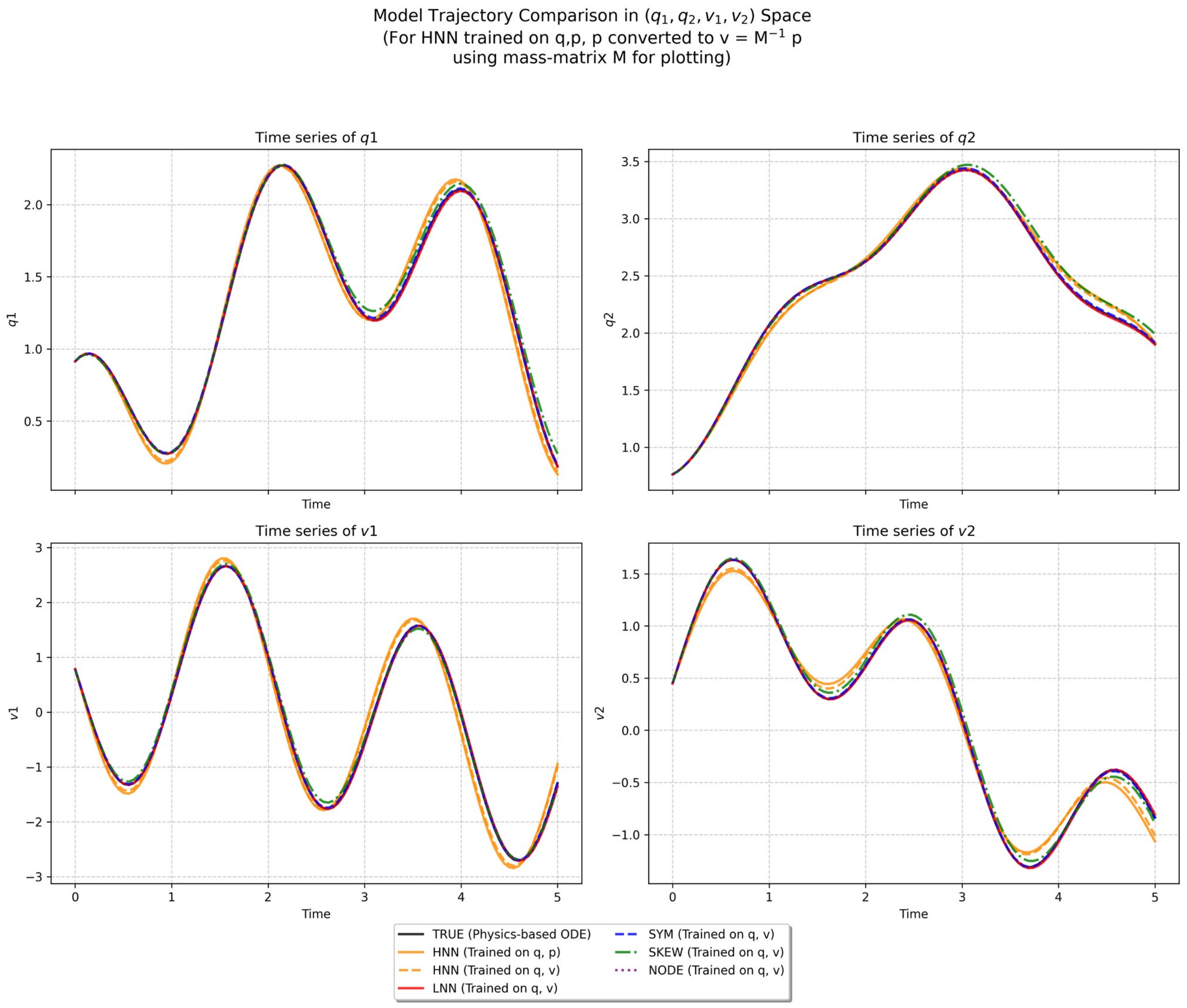


Figure 4 (a). Comparison of orbits for the mass-spring system, predicted by the NN models.

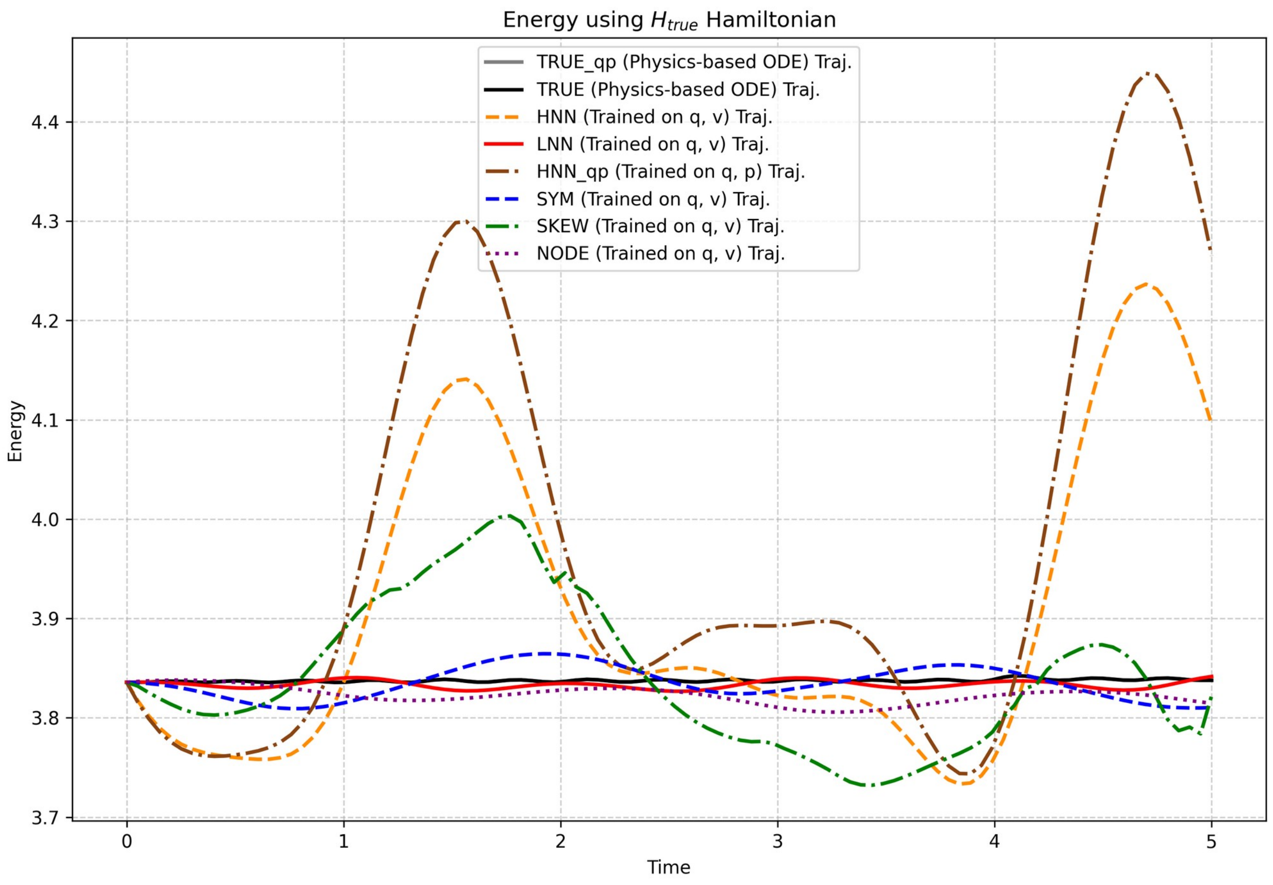


Figure 4(b). Comparison of true energy for the mass-spring system

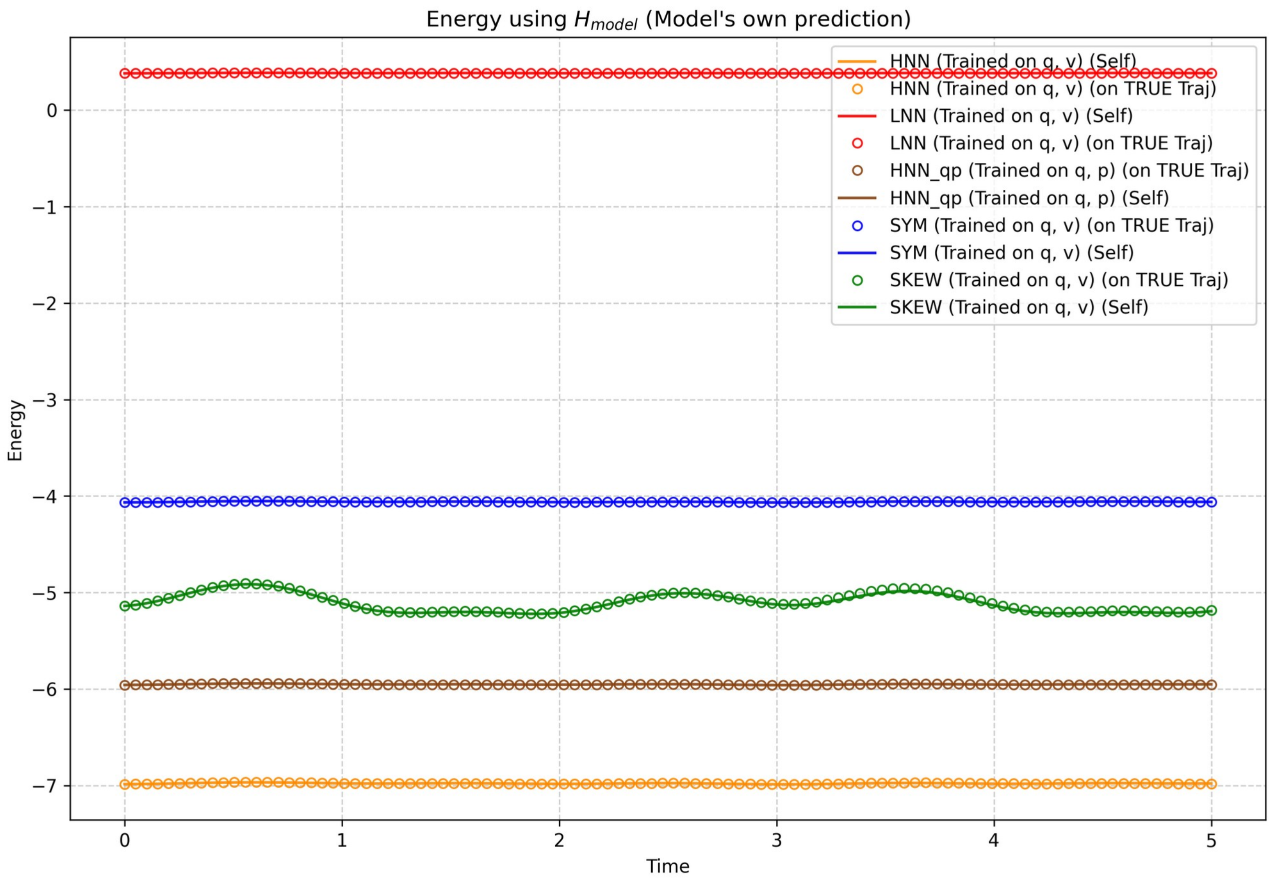


Figure 4(c). Comparison of learned energy for the mass-spring system

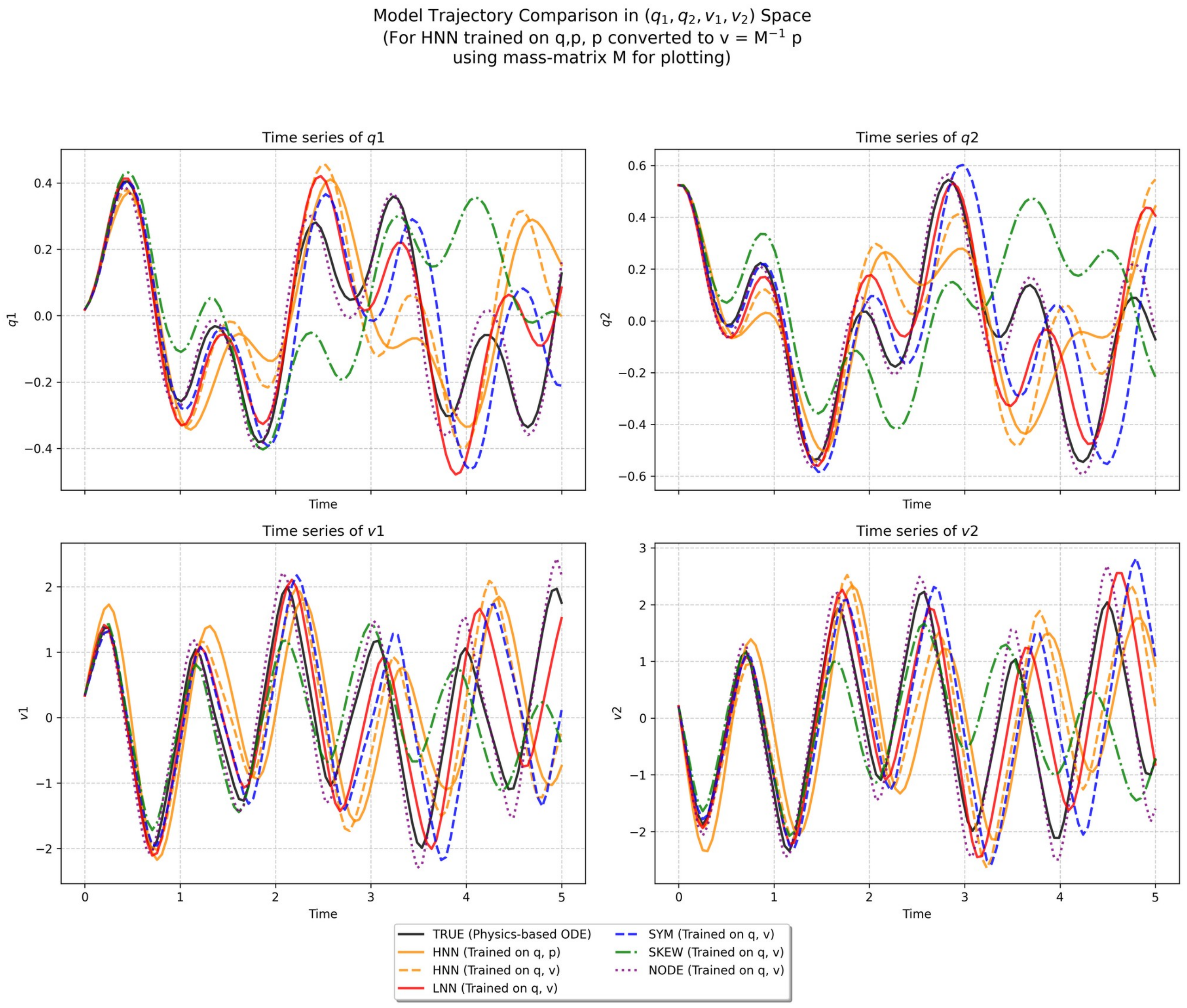


Figure 5(a). Comparison of orbits for the double-pendulum system, predicted by the NN models.

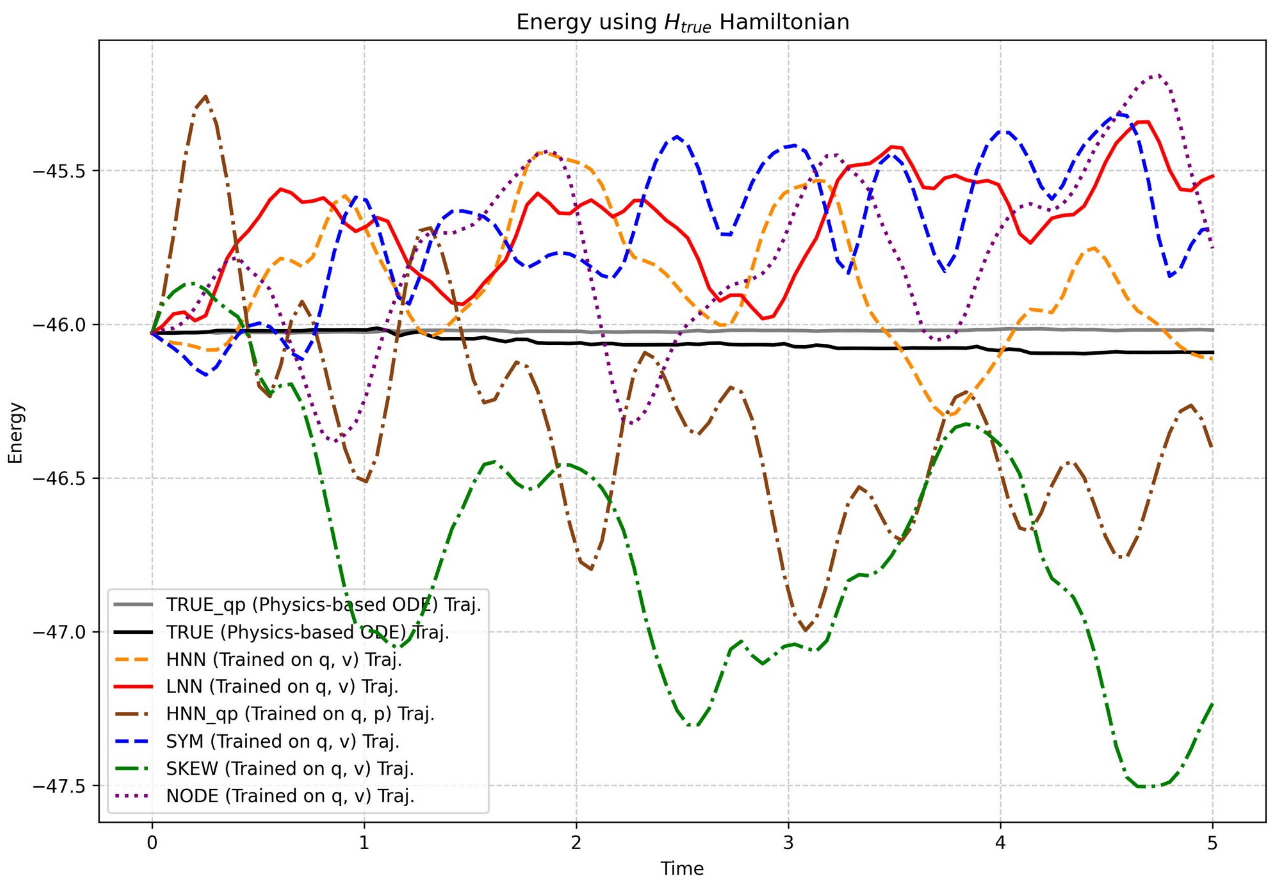


Figure 5(b). Comparison of true energy for the double-pendulum system

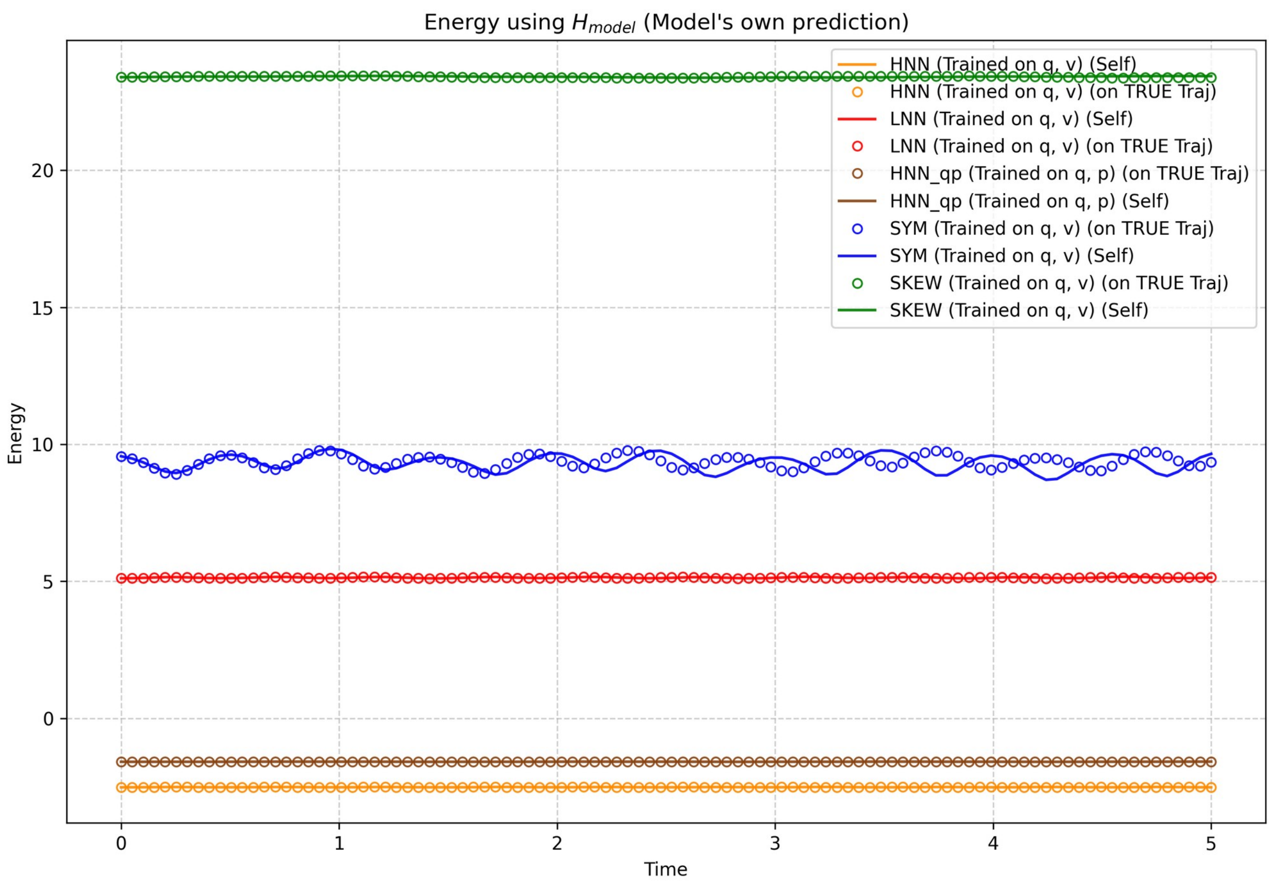


Figure 5(c). Comparison of learned energy for the double-pendulum system