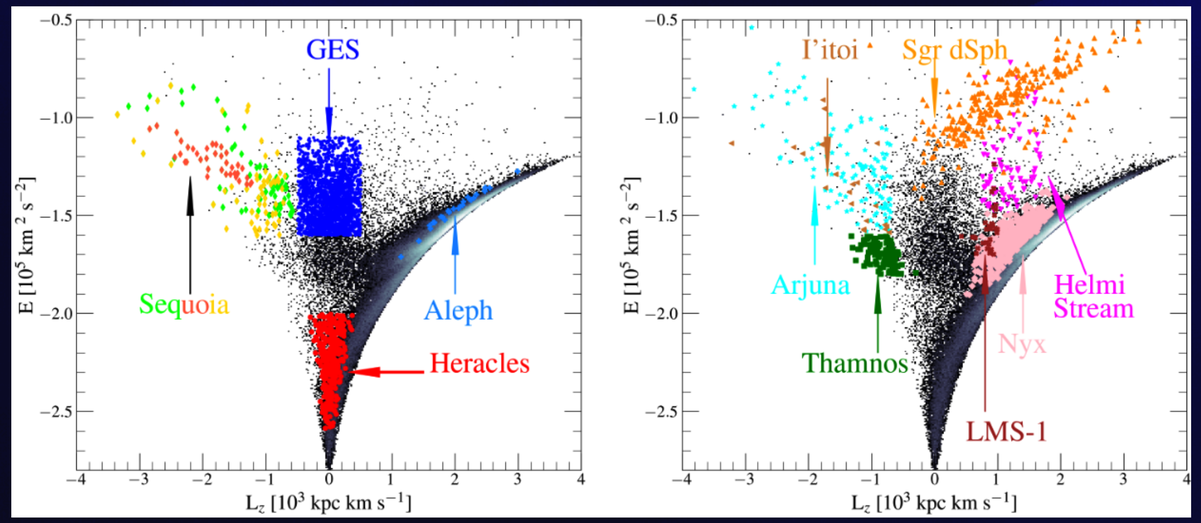
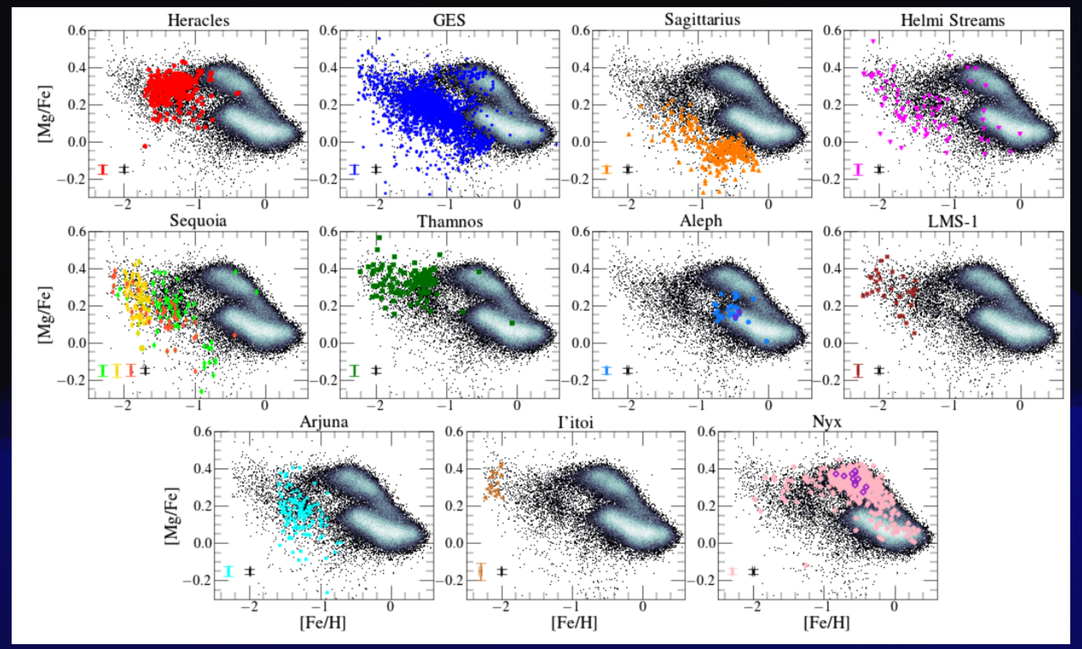
Odisseo: Optimized Differentiable Integrator for Stellar Systems Evolution of Orbit

Giuseppe Viterbo, Astro-Ai lab
Heidelberg University

Galactic Archaeology Inverse modelling

Dynamics

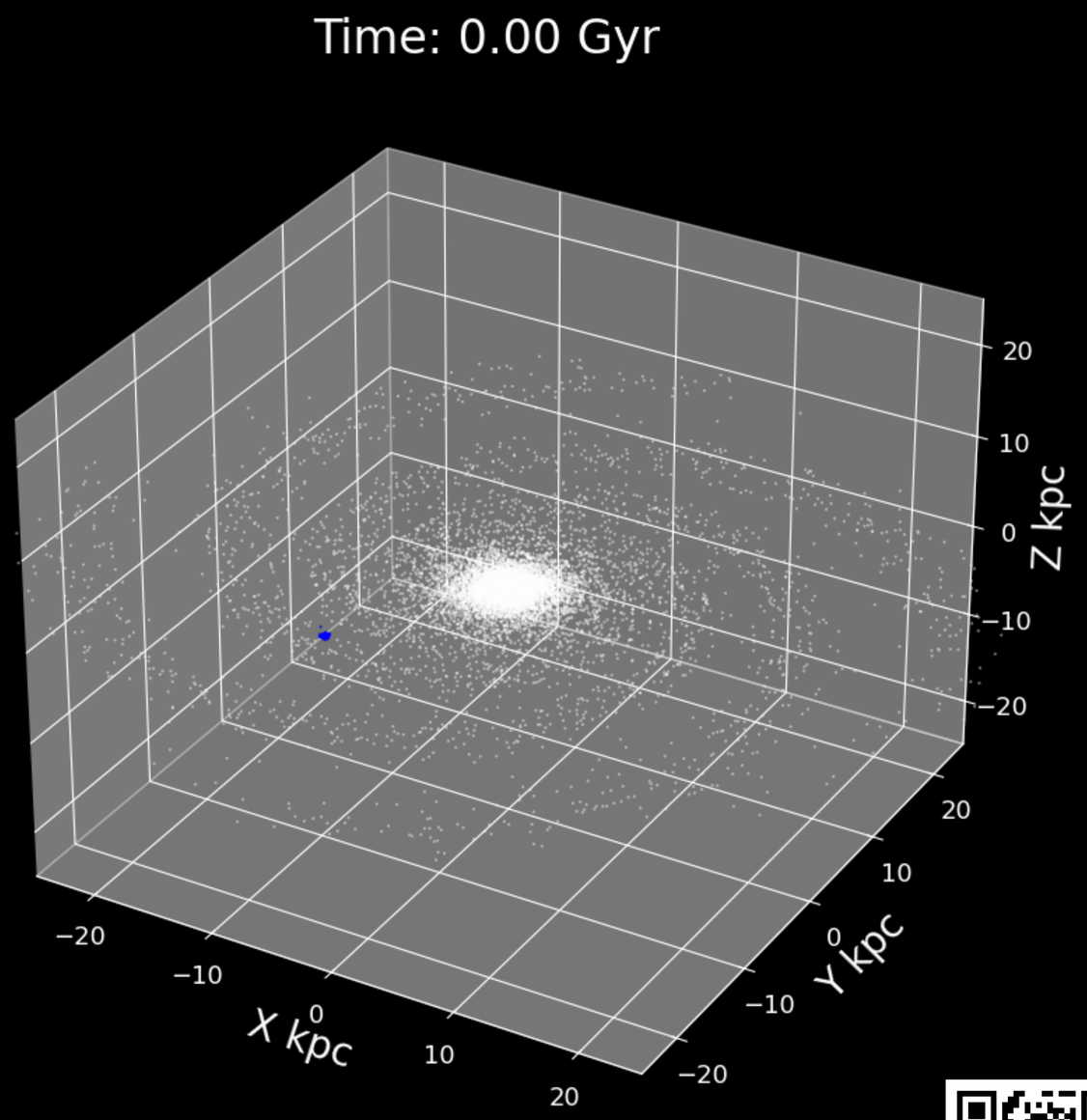




Chemistry

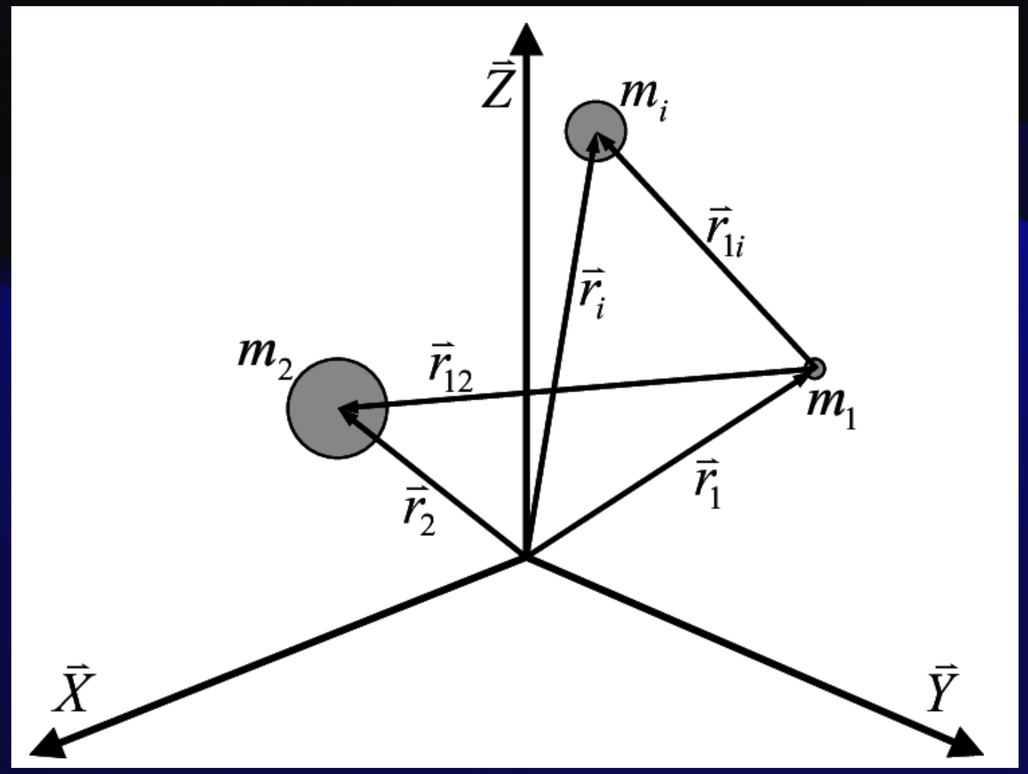
Odisseo

Differentiable direct N-body simulator

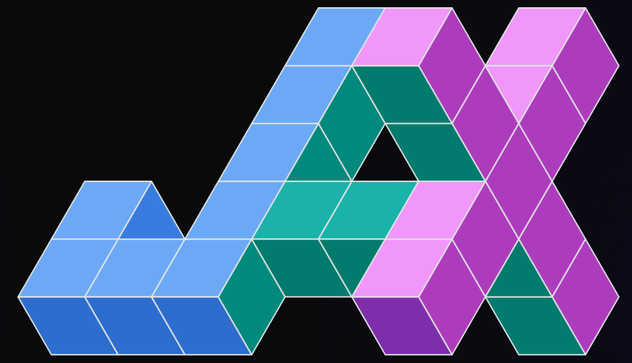


Odisseo

Differentiable direct N-body simulator

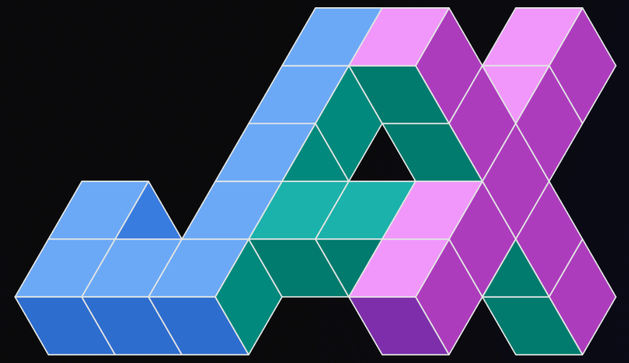


Galactic Archaeology Inverse modelling



- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !

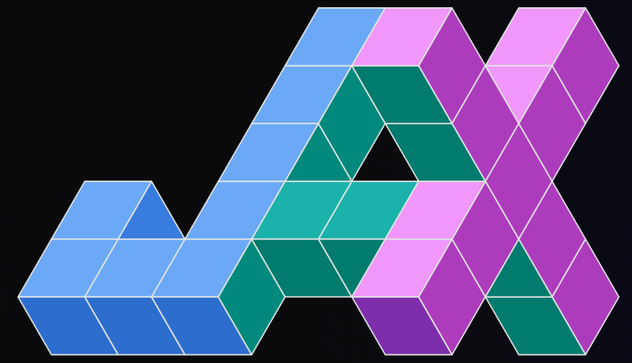
Galactic Archaeology Inverse modelling



- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !

$$Loss = -\log P(x \,|\, theta)$$
$$= -\log P(S(\theta) \,|\, \theta)$$

Galactic Archaeology Inverse modelling

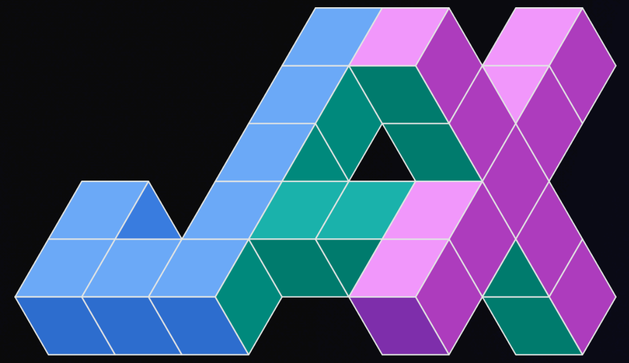


- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !

$$Loss = -\log P(x \,|\, theta)$$
$$= -\log P(S(\theta) \,|\, \theta)$$

$\longrightarrow$ Automatic-differentiation $\longrightarrow$ $\nabla_\theta \log P(x \,|\, \theta)$

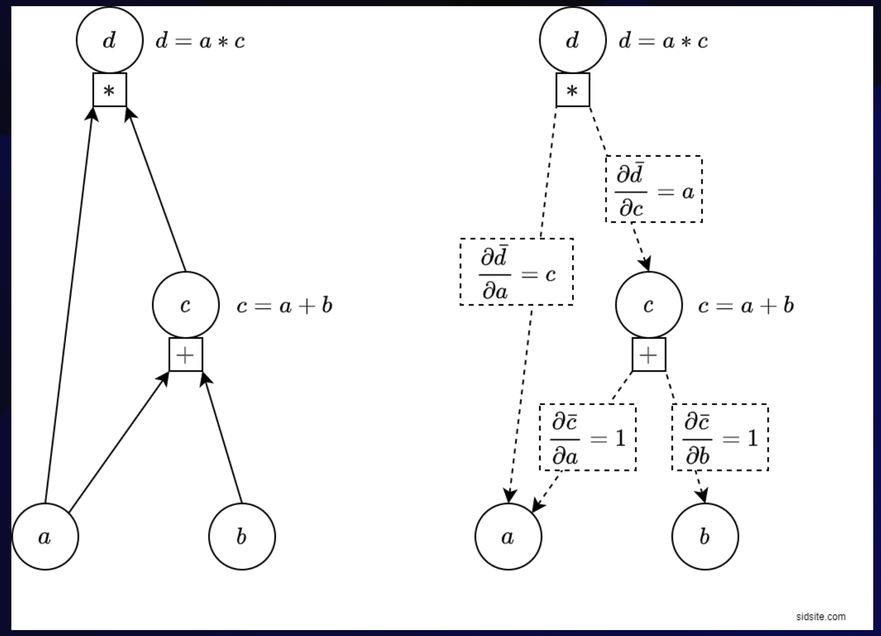
Galactic Archaeology Inverse modelling



- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !

Chain Rule!

$$Loss = -\log P(x \mid theta)$$
$$= -\log P(S(\theta) \mid \theta)$$

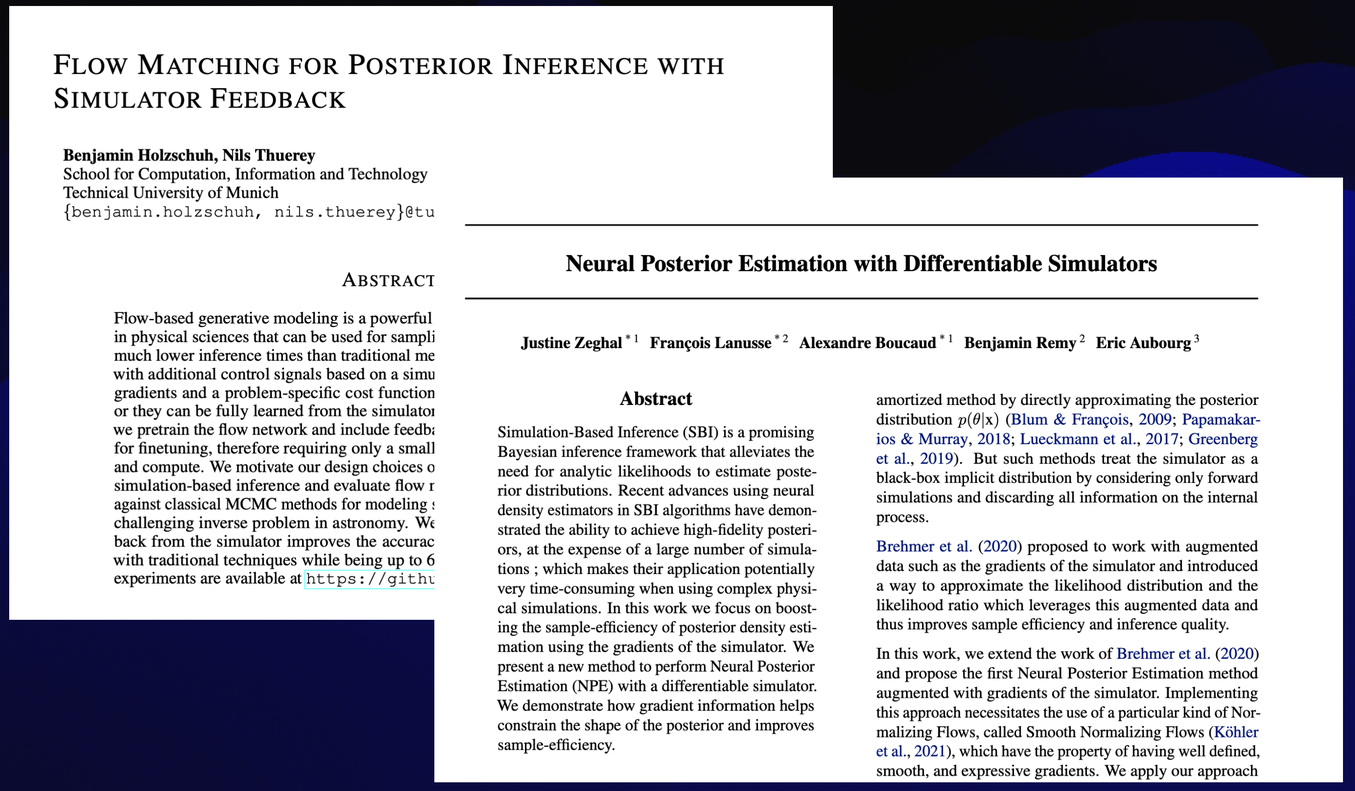


$$\nabla_\theta \log P(x \mid \theta)$$

Galactic Archaeology Inverse modelling

- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !
- The gradient can be used in many ways:
  - Direct gradient descend
  - SBI with feedback from the simulation

https://arxiv.org/pdf/2410.22573



FLOW MATCHING FOR POSTERIOR INFERENCE WITH SIMULATOR FEEDBACK

Benjamin Holzschuh, Nils Thuerey
School for Computation, Information and Technology
Technical University of Munich
{benjamin.holzschuh, nils.thuerey}@tu

ABSTRACT

Flow-based generative modeling is a powerful
in physical sciences that can be used for sampli
much lower inference times than traditional me
with additional control signals based on a simu
gradients and a problem-specific cost function
or they can be fully learned from the simulator
we pretrain the flow network and include feedb
for finetuning, therefore requiring only a small
and compute. We motivate our design choices o
simulation-based inference and evaluate flow r
against classical MCMC methods for modeling
challenging inverse problem in astronomy. We
back from the simulator improves the accurac
with traditional techniques while being up to 6
experiments are available at https://githu

Neural Posterior Estimation with Differentiable Simulators

Justine Zeghal [*1] **François Lanusse** [*2] **Alexandre Boucaud** [*1] **Benjamin Remy** [2] **Eric Aubourg** [3]

Abstract

Simulation-Based Inference (SBI) is a promising Bayesian inference framework that alleviates the need for analytic likelihoods to estimate posterior distributions. Recent advances using neural density estimators in SBI algorithms have demonstrated the ability to achieve high-fidelity posteriors, at the expense of a large number of simulations ; which makes their application potentially very time-consuming when using complex physical simulations. In this work we focus on boosting the sample-efficiency of posterior density estimation using the gradients of the simulator. We present a new method to perform Neural Posterior Estimation (NPE) with a differentiable simulator. We demonstrate how gradient information helps constrain the shape of the posterior and improves sample-efficiency.

amortized method by directly approximating the posterior distribution $p(\theta|x)$ (Blum & François, 2009; Papamakarios & Murray, 2018; Lueckmann et al., 2017; Greenberg et al., 2019). But such methods treat the simulator as a black-box implicit distribution by considering only forward simulations and discarding all information on the internal process.

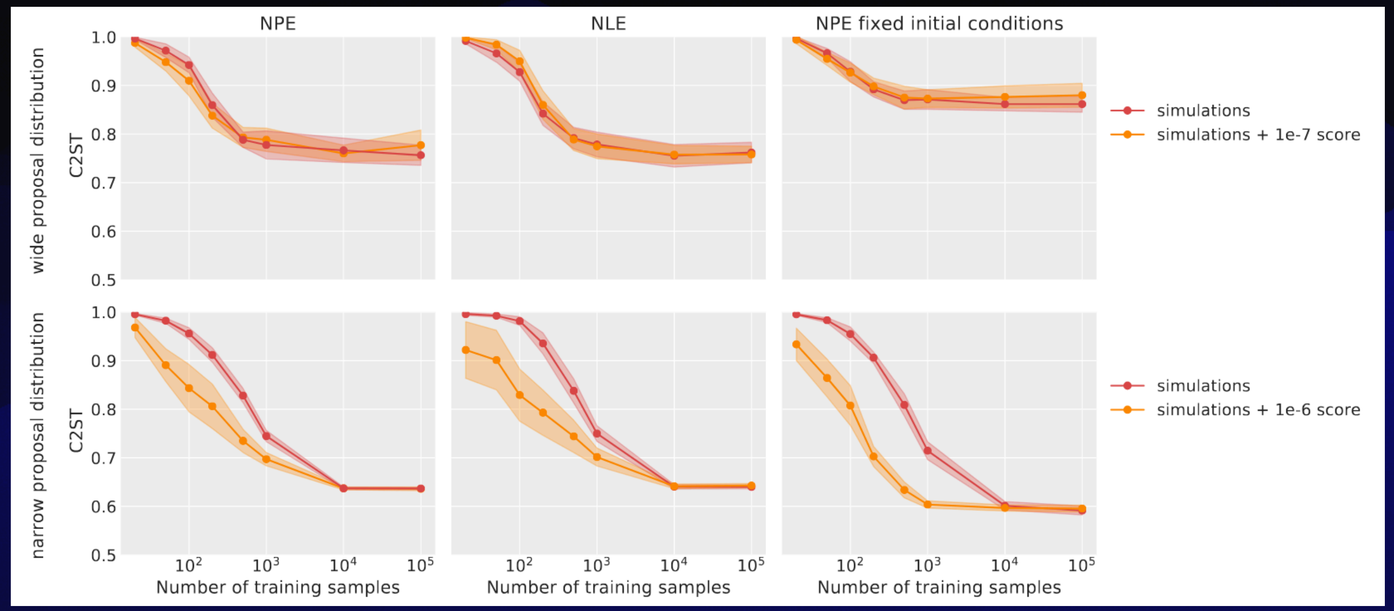
Brehmer et al. (2020) proposed to work with augmented data such as the gradients of the simulator and introduced a way to approximate the likelihood distribution and the likelihood ratio which leverages this augmented data and thus improves sample efficiency and inference quality.

In this work, we extend the work of Brehmer et al. (2020) and propose the first Neural Posterior Estimation method augmented with gradients of the simulator. Implementing this approach necessitates the use of a particular kind of Normalizing Flows, called Smooth Normalizing Flows (Köhler et al., 2021), which have the property of having well defined, smooth, and expressive gradients. We apply our approach

https://arxiv.org/abs/2207.05636

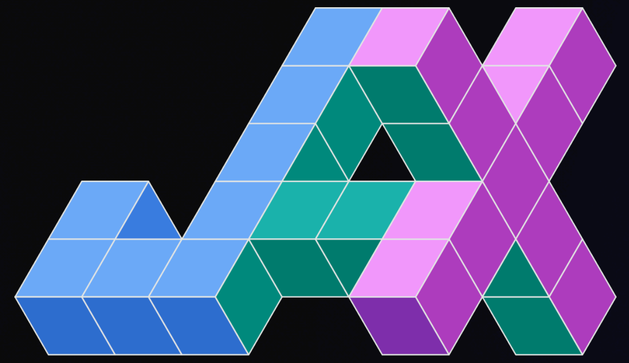
Galactic Archaeology Inverse modelling

- Force calculation is exact
- Differentiable simulations S in jax: informative for inverse modeling !
- The gradient can be used in many ways:
  - Direct gradient descend
  - SBI with feedback from the simulation



"Neural Posterior estimation with differentiable simulator", Zeghal et al. 2022

Main features:

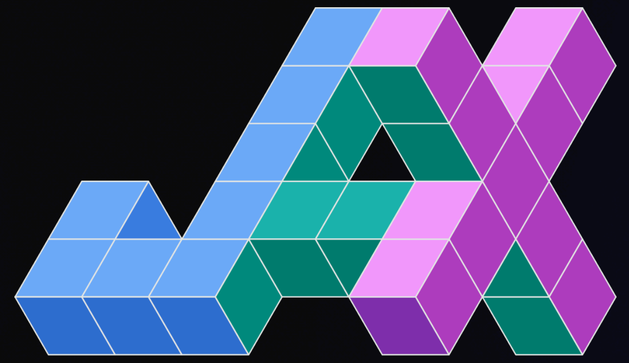


- Pure Python function:
  no side effects, deterministic,
  only jax compatible operations

```python
@partial(jax.jit, static_argnames=['config', 'return_potential'])
def direct_acc(state, mass, config, params, return_potential=False):
    """
    Compute acceleration of all particles due to all other particles by vmap of the single_body_acc function.

    Parameters
    ----------
    state : jnp.ndarray
        Array of shape (N_particles, 6) representing the positions and velocities of the particles.
    mass : jnp.ndarray
        Array of shape (N_particles,) representing the masses of the particles.
    config: NamedTuple
        Configuration parameters.
    params: NamedTuple
        Simulation parameters.

    Returns
    -------
    Tuple
        - Acceleration: jnp.ndarray
            Acecleration of all particles due to all other particles.
        - Potential: jnp.ndarray
            Potential energy of all particles due to all other particles
            Returned only if return_potential is True.

    """

    def net_force_on_body(particle_i, mass_i):

        acc, potential = vmap(lambda particle_j, mass_j: single_body_acc(particle_i, particle_j, mass_i, mass_j, config, params))(state, mass)
        if return_potential:
            return jnp.sum(acc, axis=0), jnp.sum(potential, )
        else:
            return jnp.sum(acc, axis=0)

    return vmap(net_force_on_body)(state, mass)
```

SPEED

Main features:

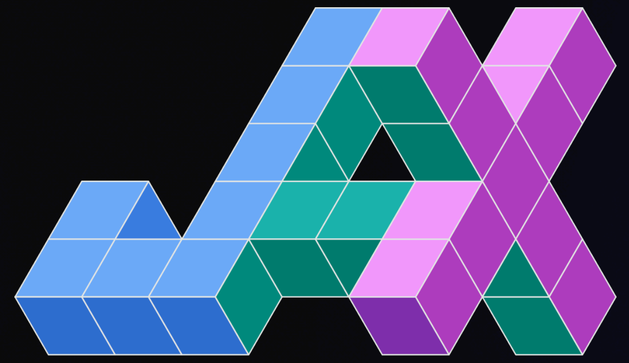


- Pure Python function:
  no side effects, deterministic,
  only jax compatible operations

```python
@partial(jax.jit, static_argnames=['config', 'return_potential'])
def direct_acc_laxmap(state, mass, config, params, return_potential=False, ):
    """
    Compute acceleration of all particles due to all other particles by vmap of the single_body_acc function.

    Parameters
    ----------
    state : jnp.ndarray
        Array of shape (N_particles, 6) representing the positions and velocities of the particles.
    mass : jnp.ndarray
        Array of shape (N_particles,) representing the masses of the particles.
    config: NamedTuple
        Configuration parameters.
    params: NamedTuple
        Simulation parameters.

    Returns
    -------
    Tuple
        - Acceleration: jnp.ndarray
            Acecleration of all particles due to all other particles.
        - Potential: jnp.ndarray
            Potential energy of all particles due to all other particles
            Returned only if return_potential is True.

    """

    def net_force_on_body(state_and_mass):
        particle_i, mass_i = state_and_mass

        if config.double_map:
            @partial(jax.jit,)
            def single_body_acc_lax(state_and_mass_j):
                particle_j, mass_j = state_and_mass_j
                return single_body_acc(particle_i, particle_j, mass_i, mass_j, config, params)
            acc, potential = jax.lax.map(single_body_acc_lax, (state, mass), batch_size=config.batch_size)
        else:
            acc, potential = vmap(lambda particle_j, mass_j: single_body_acc(particle_i, particle_j, mass_i, mass_j, config, params))(state, mass)

        if return_potential:
            return jnp.sum(acc, axis=0), jnp.sum(potential, )
        else:
            return jnp.sum(acc, axis=0)

    return jax.lax.map(net_force_on_body, (state, mass), batch_size=config.batch_size)
```

Memory efficient

Main features:



- Pure Python function:
  no side effects, deterministic,
  only jax compatible operations
- Easy to set config and params

Modular

```
# Define the
config = SimulationConfig(N_particles=10_000,
                          return_snapshots=True,
                          num_snapshots=100,
                          num_timesteps=1000,
                          external_accelerations=(NFW_POTENTIAL,  ),
                          acceleration_scheme=DIRECT_ACC,
                          softening=(0.1 * u.kpc).to(code_units.code_length).value) #default values

params = SimulationParams(t_end = (10 * u.Gyr).to(code_units.code_time).value,
                          Plummer_params= PlummerParams(Mtot=(1e8 * u.Msun).to(code_units.code_mass).value,
                                                        a=(1 * u.kpc).to(code_units.code_length).value),
                          NFW_params = NFWParams(Mvir=(1e12 * u.Msun).to(code_units.code_mass).value,
                                                 r_s = (20 * u.kpc).to(code_units.code_length).value,
                                                 c = 10,),
                          G=G, )
```

Main features:

- Pure Python function:
  no side effects, deterministic,
  only jax compatible operations
- Easy to set config and params
- Astropy units

```
code_length = 10.0 * u.kpc
code_mass = 1e8 * u.Msun
G = 1
code_units = CodeUnits(code_length, code_mass, G=G)
```

Main features:

- Pure Python function:
  no side effects, deterministic,
  only jax compatible operations
- Easy to set config and params
- Astropy units
- Multiple external potentials

```python
@partial(jax.jit, static_argnames=['config', 'return_potential'])
def MyamotoNagai(state, config, params, return_potential=False):
    @partial(jax.jit, static_argnames=['config', 'return_potential'])
    def point_mass(state, config, params, return_potential=False):
        @partial(jax.jit, static_argnames=['config', 'return_potential'])
        def NFW(state, config, params, return_potential=False):
            """
            Compute acceleration of all particles due to a NFW profile.

            Parameters
            ----------
            state : jnp.ndarray
                Array of shape (N_particles, 6) representing the positions and velocities of the particles.
            config: NamedTuple
                Configuration parameters.
```

Visualization

Power Spherical Potential Cutoff

$$\rho(R) = -\left(\frac{r_s}{R}\right)^{\alpha} \exp\left(-\left(\frac{R}{r_c}\right)^2\right)$$

NFW

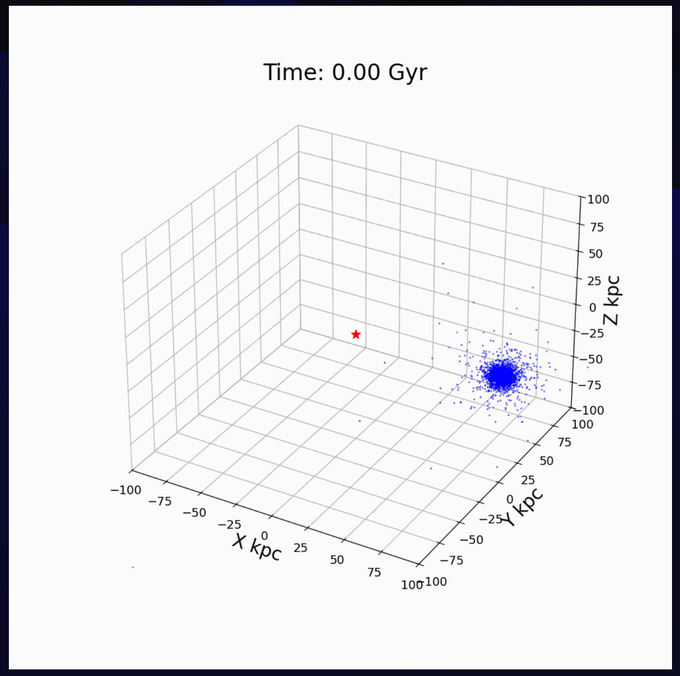
$$\Phi(r) = -\frac{4\pi G \rho_0 R_s^3}{r} \ln\left(1 + \frac{r}{R_s}\right)$$

Miyamoto Nagai

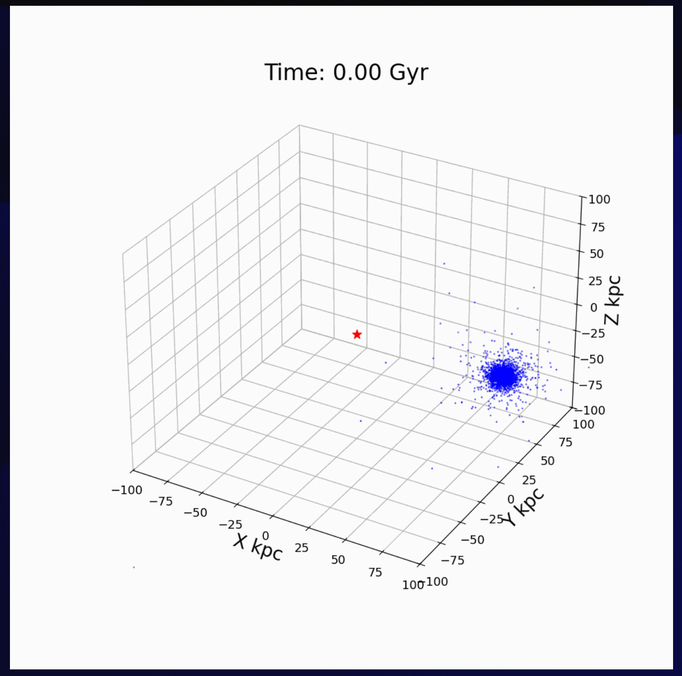
$$\Phi(R, z) = -\frac{GM}{\sqrt{R^2 + (\sqrt{z^2 + b^2} + a)^2}}$$

Visualization

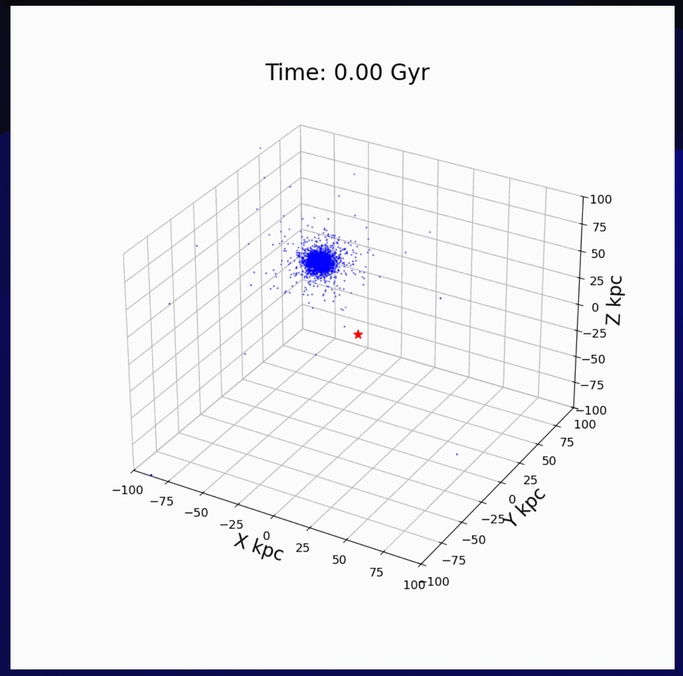
Power Spherical Potential Cutoff



NFW

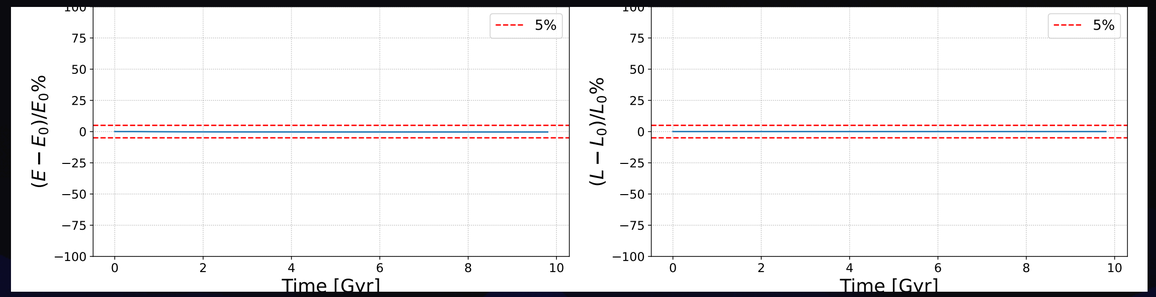


Miyamoto Nagai

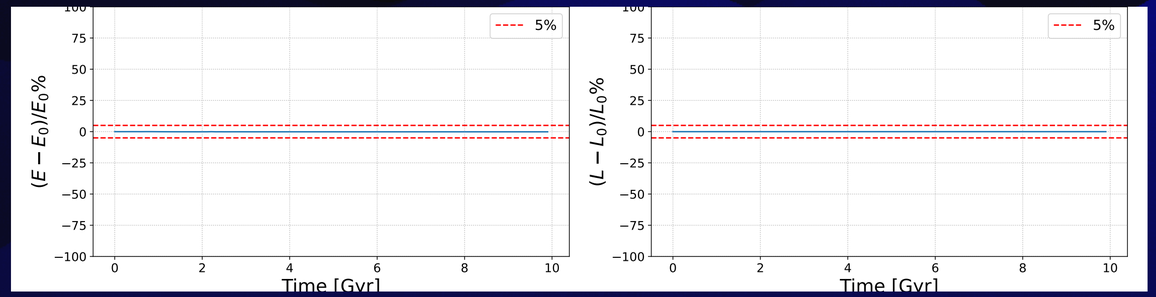


Conserved property

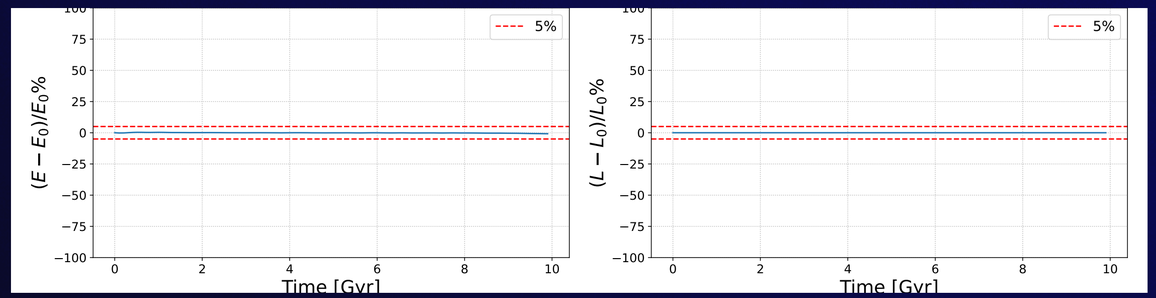
PointMass



NFW



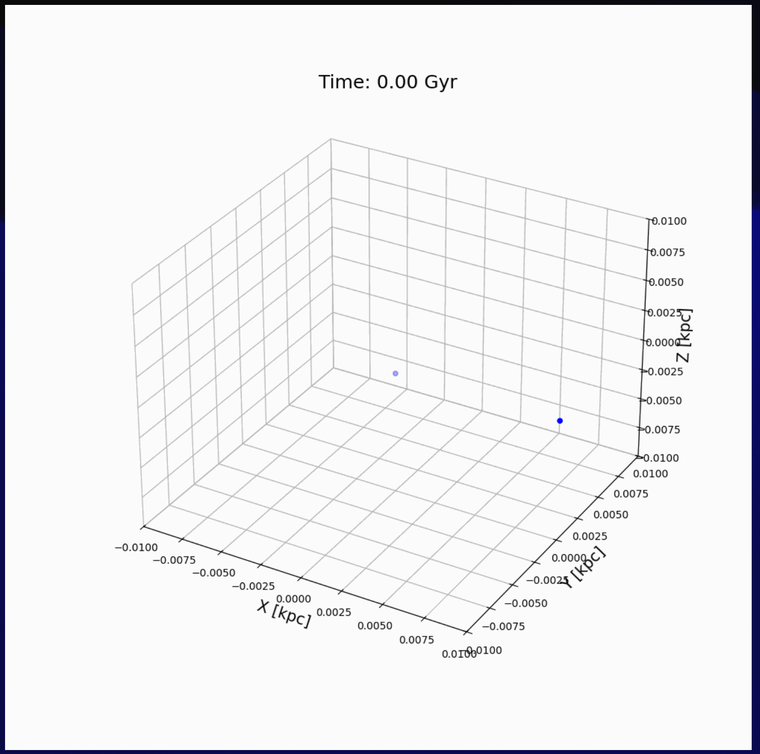
Miyamoto Nagai



Main features:

- Initial conditions

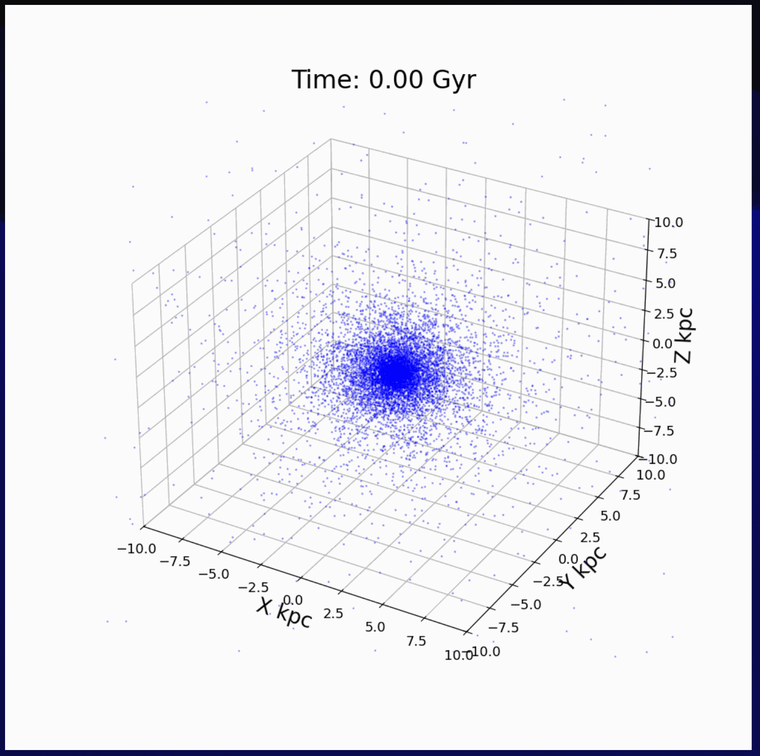
```python
@partial(jax.jit)
def ic_two_body(mass1: Union[float, jnp.ndarray],
                mass2: Union[float, jnp.ndarray],
                rp: Union[float, jnp.ndarray],
                e: Union[float, jnp.ndarray],
                params: SimulationParams) -> Tuple:
    """
    Create initial conditions for a two-body system.

    By default, the two bodies will be placed along the x-axis at the
    closest distance rp. Depending on the input eccentricity, the two
    bodies can be in a circular (e < 1), parabolic (e = 1), or hyperbolic
    orbit (e > 1).
```



Main features:

- Initial conditions

```python
@jaxtyped(typechecker=typechecker)
@partial(jax.jit, static_argnames=['config'])
def Plummer_sphere(key: PRNGKeyArray,
                   config: SimulationConfig,
                   params: SimulationParams) -> Tuple:
    """
    Create initial conditions for a Plummer sphere. The sampling of velocities
    is done by inverse fitting the cumulative distribution function of the
    Plummer sphere.      You, 5 days ago • Uncommitted changes
```

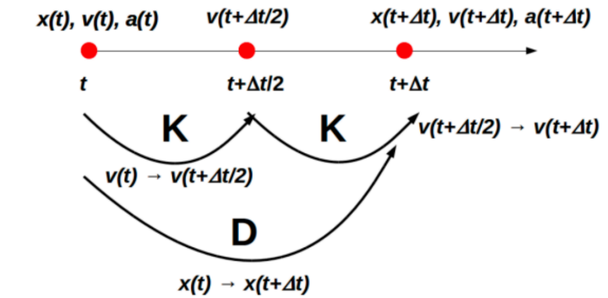


Main features:

- Initial conditions
- Integrator(s):
  - Leap-frog
  - RK4

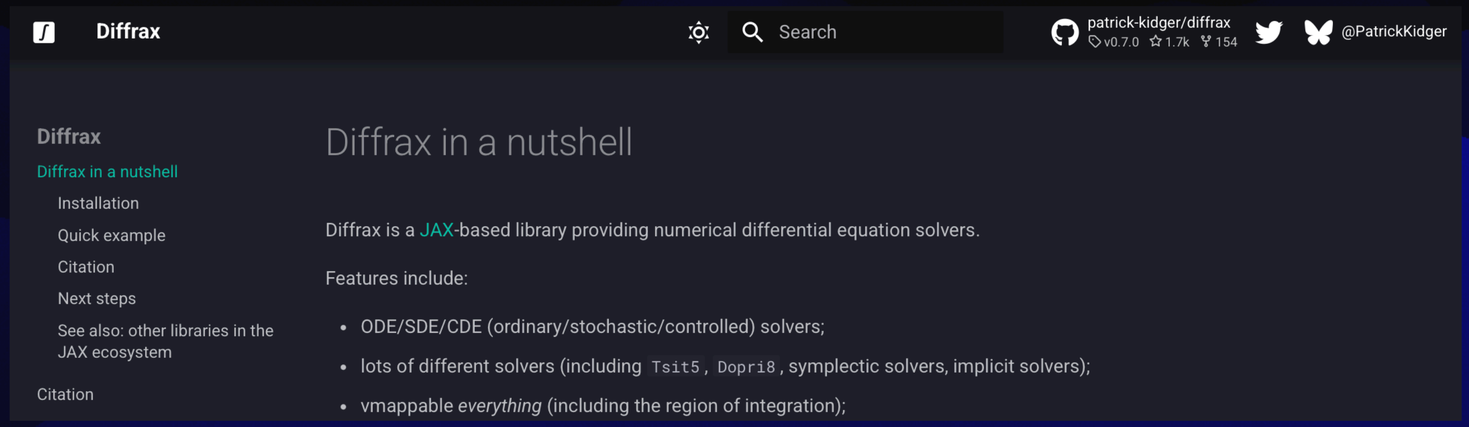
```python
@partial(jax.jit, static_argnames=['dt', 'config'])
def leapfrog(state, mass, dt, config, params):
    """
    Simple implementation of a symplectic Leapfrog (Verlet) integrator for N-body simulations.

    Parameters
    ----------
    state : jax.numpy.ndarray
        The state of the particles, where the first column represents positions and the second column represents velocities.
    mass : jax.numpy.ndarray
        The mass of the particles.
    dt : float
        Time-step for current integration.
    config : object
        Configuration object containing the acceleration scheme and external accelerations.
    params : dict
        Additional parameters for the acceleration functions.

    Returns
    -------
    jax.numpy.ndarray
        The updated state of the particles.
    """
    if config.acceleration_scheme == DIRECT_ACC:
        acc_func = direct_acc
            You, last month • first commit, the pipeline is running, need to …
    elif config.acceleration_scheme == DIRECT_ACC_LAXMAP:
        acc_func = direct_acc_laxmap

    acc = acc_func(state, mass, config, params)

    # Check additional accelerations
    if len(config.external_accelerations) > 0:
        acc = acc + combined_external_acceleration_vmpa_switch(state, config, params)

    # removing half-step velocity
    state = state.at[:, 0].set(state[:, 0] + state[:, 1]*dt + 0.5*acc*(dt**2))

    acc2 = acc_func(state, mass, config, params)

    if len(config.external_accelerations) > 0:
        acc2 = acc2 + combined_external_acceleration_vmpa_switch(state, config, params)

    state = state.at[:, 1].set(state[:, 1] + 0.5*(acc + acc2)*dt)

    return state
```

$$\vec{r}_{i+1} = \vec{r}_i + \vec{v}_i h + \frac{1}{2}\vec{a}_i h^2$$
$$\vec{a}_{i+1} = f(\vec{r}_{i+1})$$
$$\vec{v}_{i+1} = \vec{v}_i + \frac{1}{2}\left(\vec{a}_i + \vec{a}_{i+1}\right) h$$



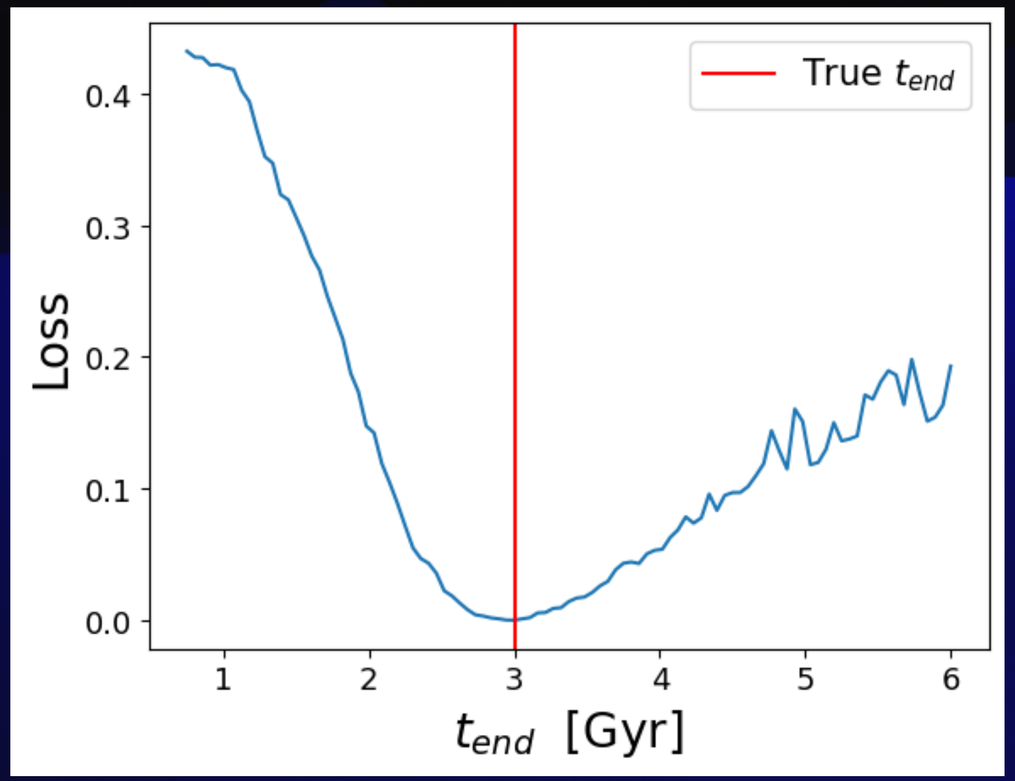
Main features:

- Initial conditions
- Integrator(s):
  - Leap-frog
  - RK4
  - Diffrax backend



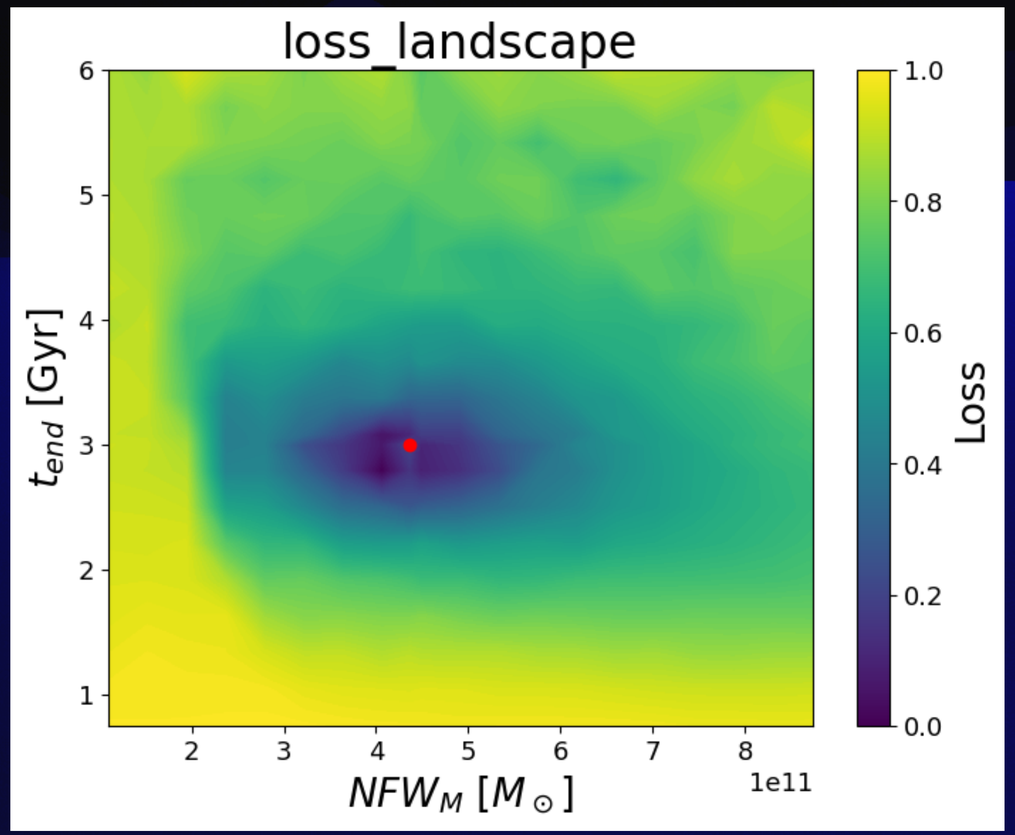
Main features:

- Initial conditions
- Integrator(s)
- Gradient! (WIP)
  - 1 parameter



Main features:

- Initial conditions
- Integrator(s)
- Gradient! (WIP)
  - 1 parameter
  - 2 parameters
  - Full inference and SBI integration !



Thank you for your attention

Summary:

- Differentiable N-body code
- Jax friendly
- GitHub repo: https://github.com/vepe99/Odisseo
- Qr code:

ROAD MAP:

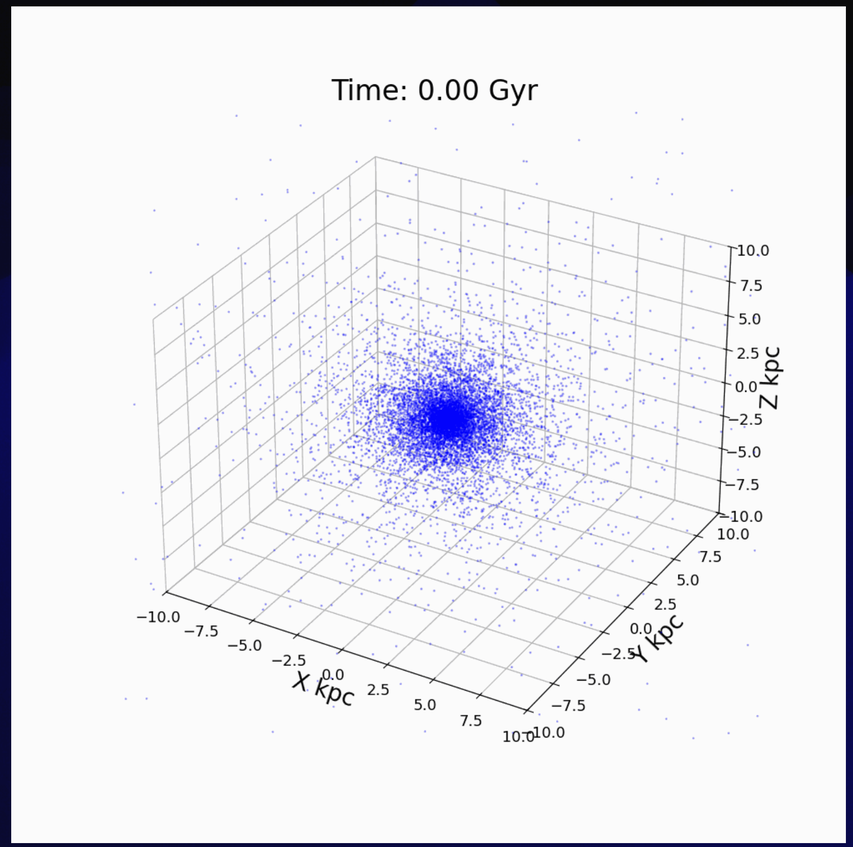
REBOUND

- More integrators !
  - IAS15
  - WHFast512

ROAD MAP:

REBOUND

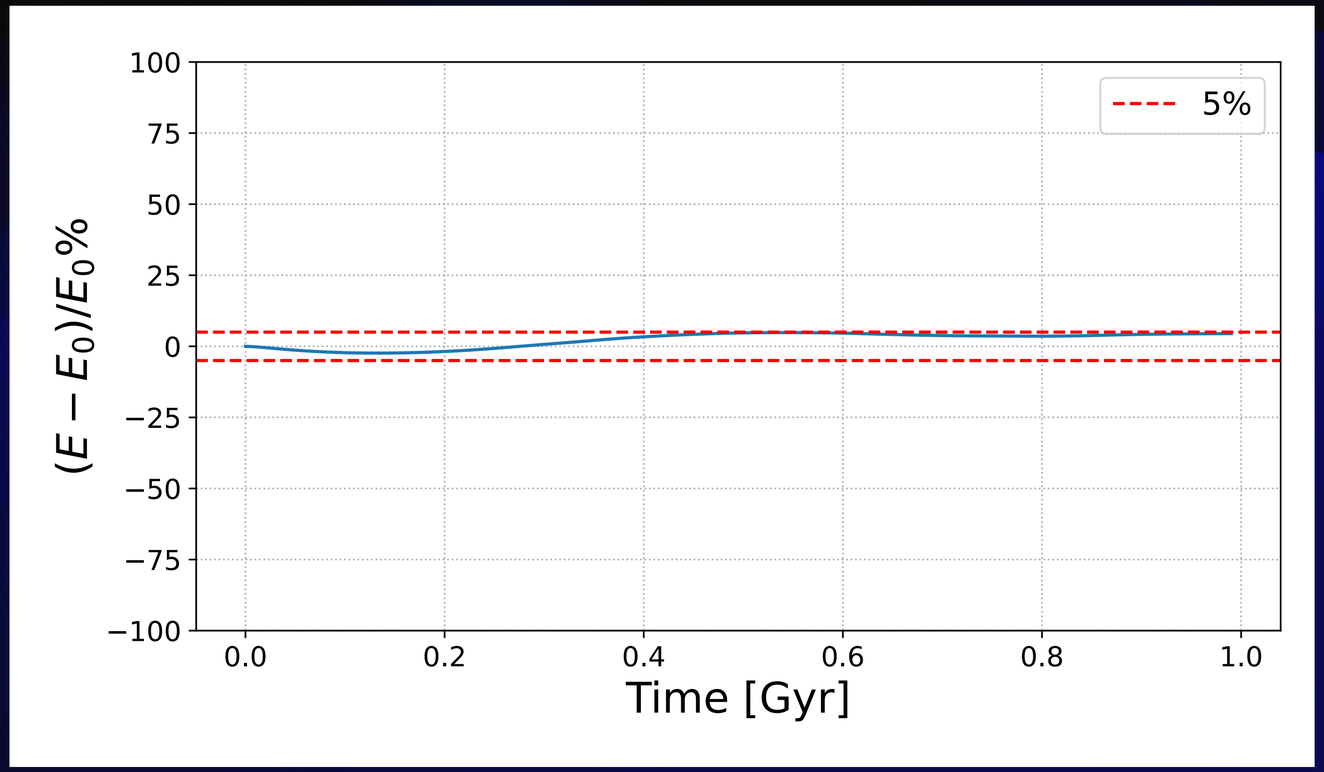
- More integrators !
  - IAS15
  - WHFast512



ROAD MAP:

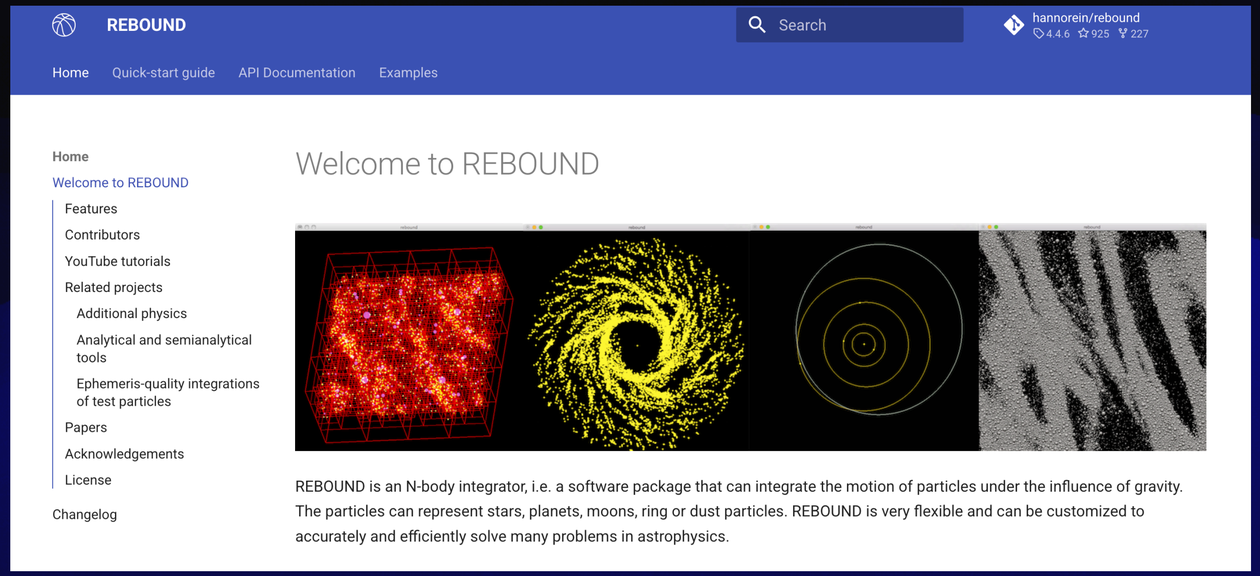
REBOUND

- More integrators !
  - IAS15
  - WHFast512



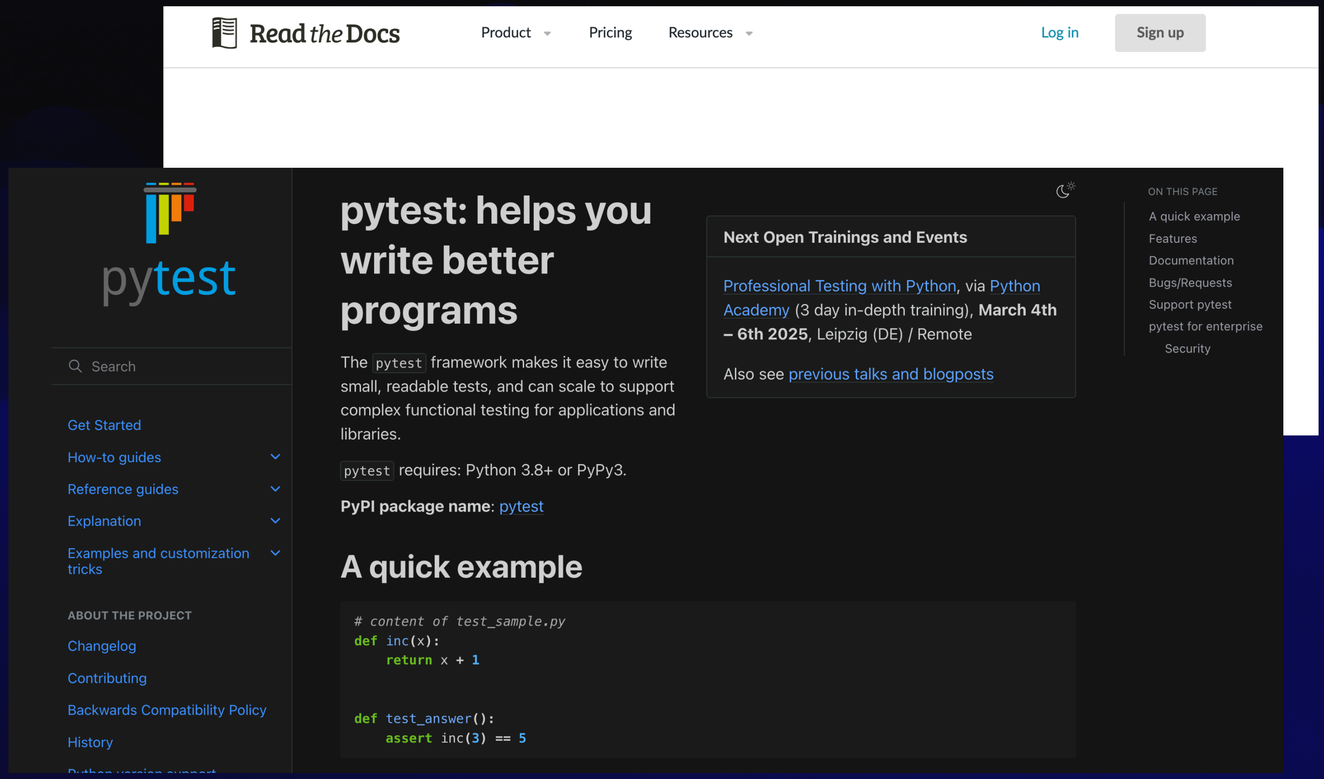
ROAD MAP:

- More integrators
- Evaluate the accuracy with other Nbody code



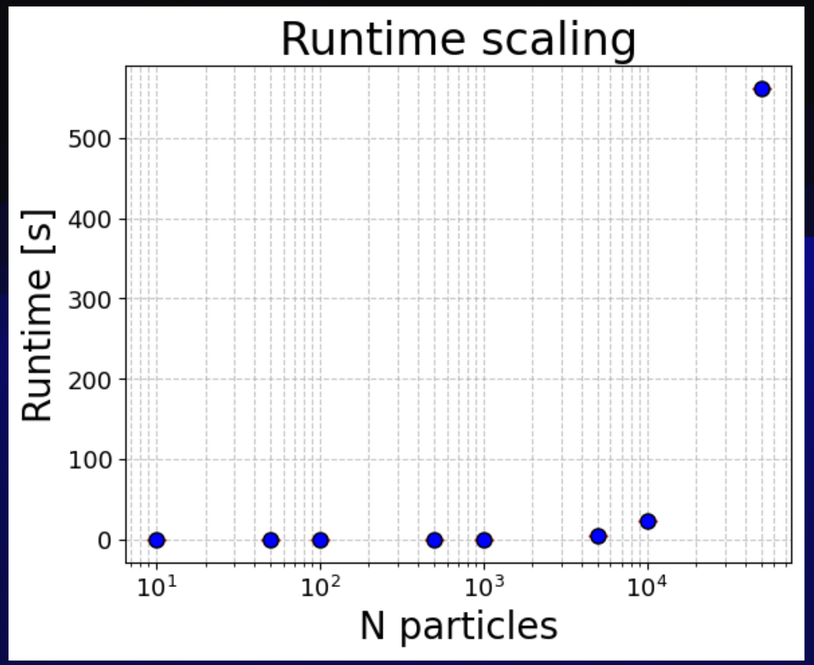
ROAD MAP:

- More integrators
- Evaluate the accuracy with other Nbody code
- Testing, documentation, …



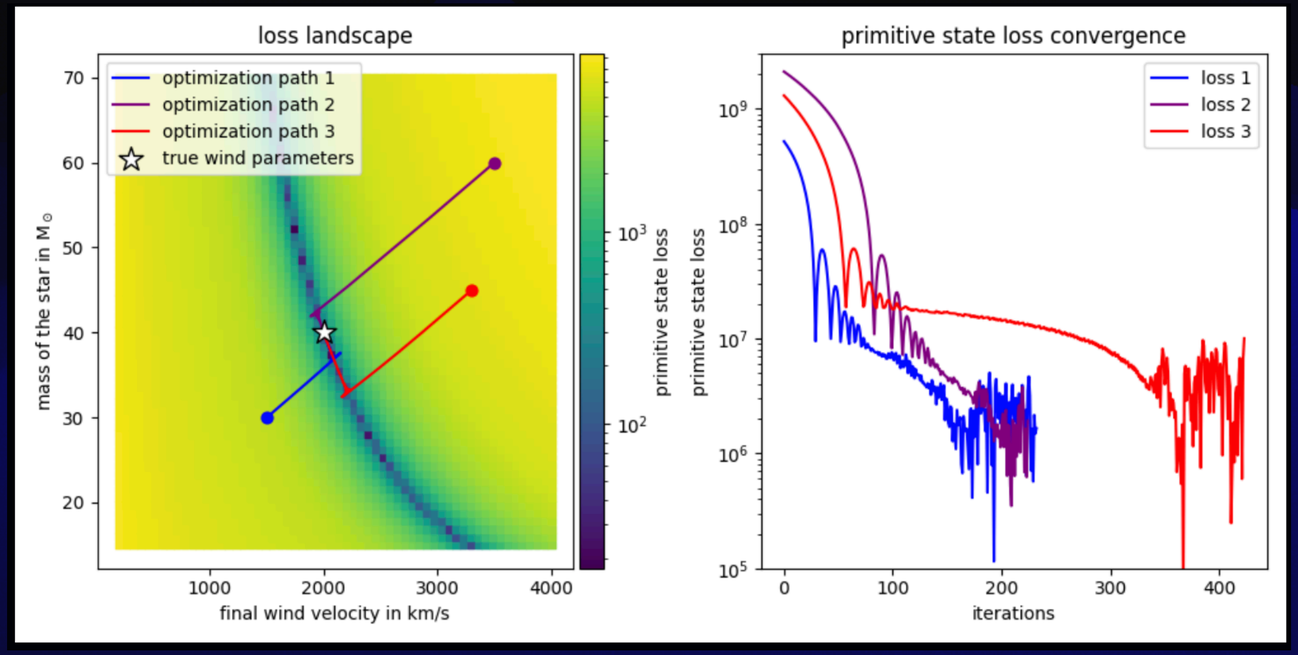
ROAD MAP:

- More integrators
- Evaluate the accuracy with other Nbody code
- Testing, documentation, …
- Scalability (speed and memory)



ROAD MAP:

- More integrators
- Evaluate the accuracy with other Nbody code
- Testing, documentation, …
- Scalability (speed and memory)
- Loss function/Gradient Optimization



“Differentiable Conservative Radially Symmetric Fluid Simulations and Stellar Winds ○ jf1uids”, Storcks L. And Buck T. 2024

ROAD MAP:

- More integrators
- Evaluate the accuracy with other Nbody code
- Testing, documentation, …
- Scalability (speed and memory)
- Loss function/Gradient Optimization
- SBI (finally!)

Additional slides

Sampling of the Plummer potential

- Initial conditions

Initial Conditions for self-gravitating systems

Initial positions for a spherical system, Plummer sphere

The Plummer density model often provides a good description of the mass distribution in stellar systems (see Lecture 3)

$$\rho(r) = -\frac{3M_{\rm tot}}{4\pi a^3}\left(1+\frac{r^2}{a^2}\right)^{-\frac{5}{2}} \qquad M(r) = M_{\rm tot}\left(\frac{r}{a}\right)^3\left(1+\frac{r^2}{a^2}\right)^{-\frac{3}{2}} \qquad M_{\rm tot} = M(\infty)$$

So:

$$F(r) = \frac{M(r)}{M_{\rm tot}} = \left(\frac{r}{a}\right)^3\left(1+\frac{r^2}{a^2}\right)^{-\frac{3}{2}} = u$$

The equation can be inverted so that, we can sample the radius with:

$$r = \sqrt{\frac{a^2}{u^{-\frac{2}{3}}-1}} \qquad \text{where } u \sim \mathcal{U}(0,1)$$

“Computational Astrophysics lecture slide”, Giuliano Iorio

Sampling of the Plummer potential

- Initial conditions

For a stationary, spherical, isotropic system the df depend only on the specific energy of the particle $f(\tilde{E}_{tot})$

Initial Conditions for self-gravitating systems

Initial velocity for a spherical and isotropic system - Distribution function

For a Plummer sphere,

$$f(\tilde{E}_{\rm tot}) = \begin{cases} -A\tilde{E}_{\rm tot}^{\frac{7}{2}} & \text{if} \quad \tilde{E} \leq 0 \\ 0 & \text{if} \quad \tilde{E} > 0 \end{cases} \qquad \Phi_P(r) = -\frac{GM_{\rm tot}}{\sqrt{r^2 + a^2}}$$

At a given radius, we know the Plummer potential $\Phi_P(\vec{r})$, so f ranges from 0 (when $|\vec{v}|^2 = 2\Phi_P(\vec{r})$) to the maximum $f(\tilde{E}_{\rm tot} = \Phi_P(\vec{r})) = f_{\rm max}(\vec{r})$.

So, we can use the modified df:

$$\tilde{f} = f(\tilde{E}_{\rm tot})/f_{\rm max}(\vec{r}) = \left(\frac{\tilde{E}_{\rm tot}}{\Phi_P(\vec{r})}\right)^{\frac{7}{2}}$$

“Computational Astrophysics lecture slide”, Giuliano Iorio

Sampling of the Plummer potential

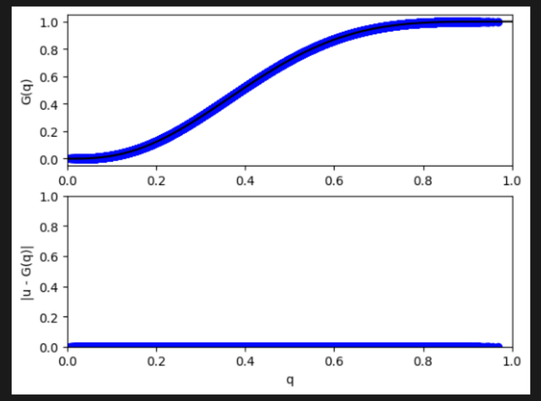
- Initial conditions

Inverse cdf interpolation

```
fig = plt.figure()

def G(q, ):
    return 1287/16 * ((-2*(1-q)**(9/2))*(99*q**2+36*q+8)/1287 +16/1287)

q = jnp.linspace(0, 1, num=10_000_000)
y = G(q, )

ax = fig.add_subplot(211)
ax.plot(q, y, color='black')

u = random.uniform(key=random.PRNGKey(0), shape=(100_000,))
sample = jnp.interp(u, y, q)
ax.scatter(sample, u, color='blue')
ax.set_ylabel('G(q)')
ax.set_xlim(0, 1)

ax = fig.add_subplot(212)
ax.scatter(sample, abs(u-G(sample)), color='blue')
ax.set_xlabel('q')
ax.set_ylabel('|u - G(q)|')
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
```

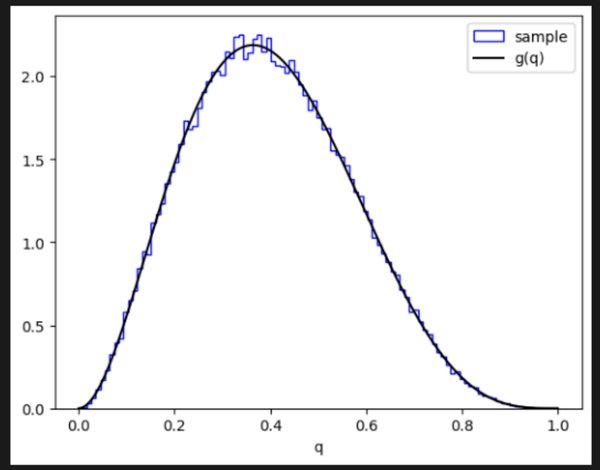
[18]

```
(0.0, 1.0)
```



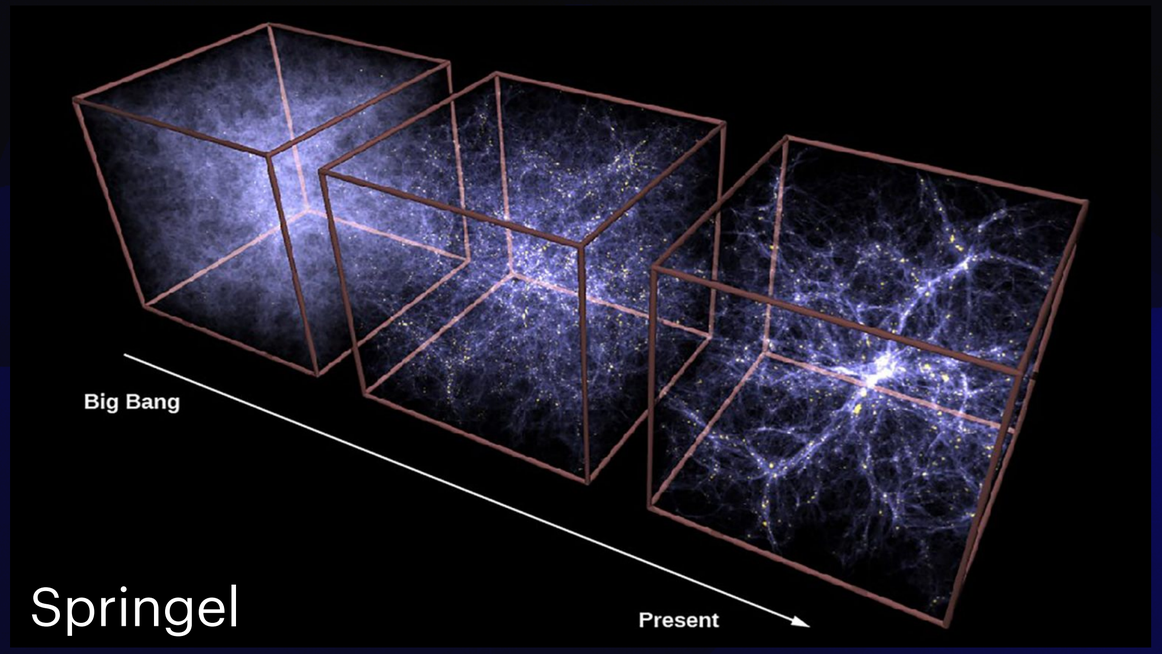

```
plt.hist(sample, histtype='step', color='blue',bins=100, density=True, label='sample')
q = jnp.linspace(0, 1, num=100_000)
y = 1287/16*((1-q)**(7/2))*q**2
plt.plot(q, y, color='black', label='g(q)')
plt.xlabel('q')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x16b1a4890>
```



Hierarchical growth

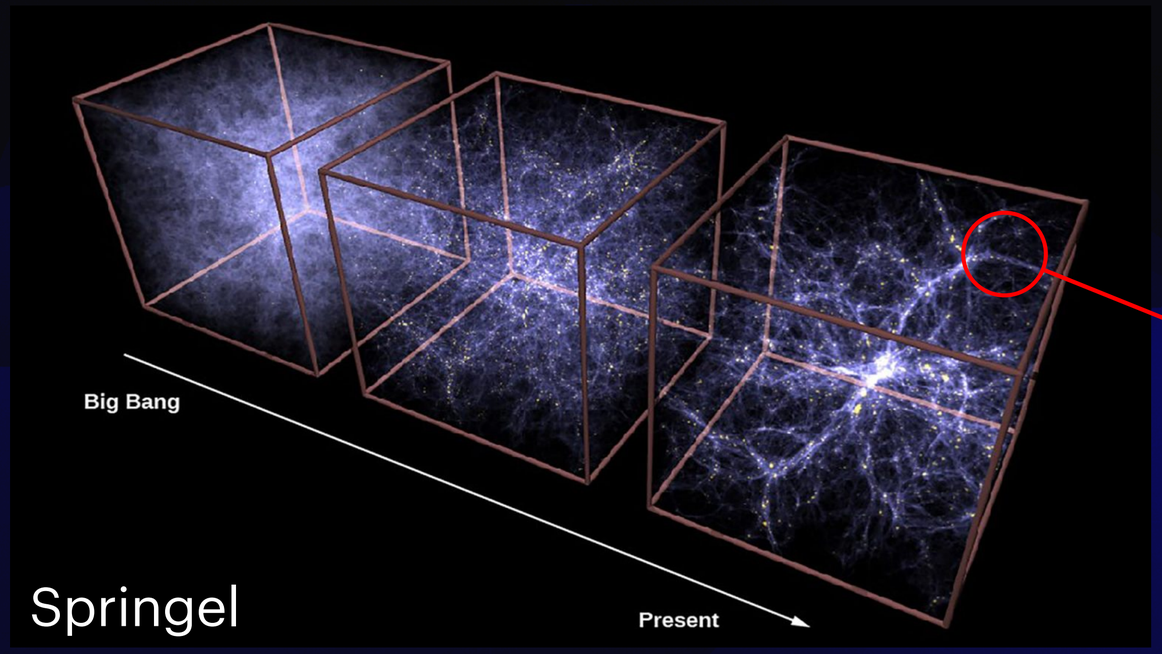
Hierarchical growth



$$\Lambda CDM : \frac{dn}{dM} \propto M^{-1.9}$$

‘Exploring the Geometry, Topology and Morphology of Large Scale Structur using Minkowski Functionals’, Sheth et al. 2005.

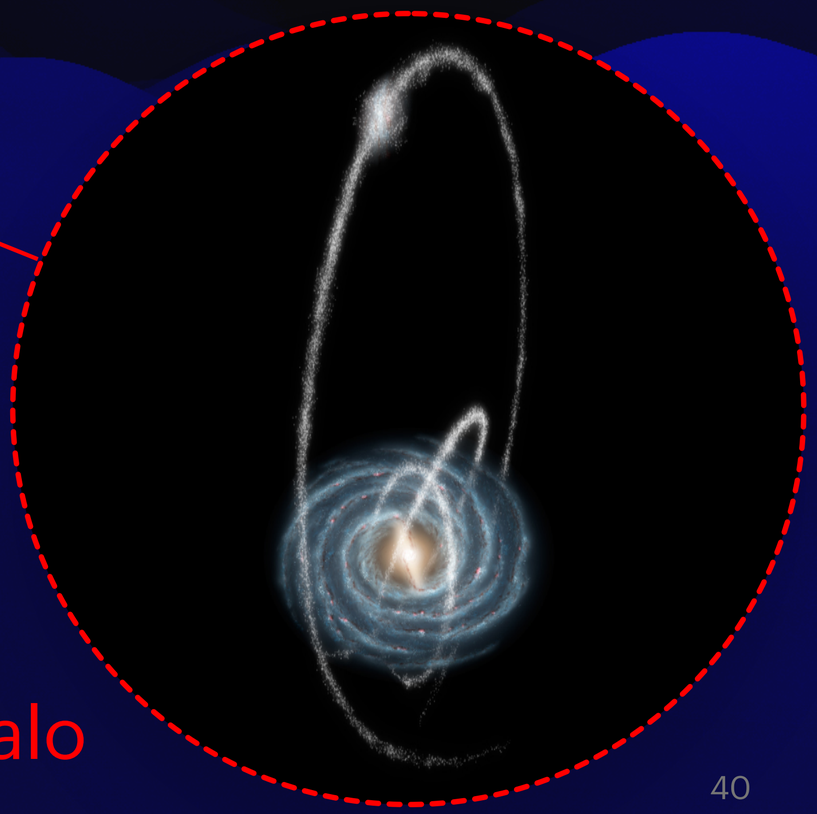
Hierarchical growth



Springel

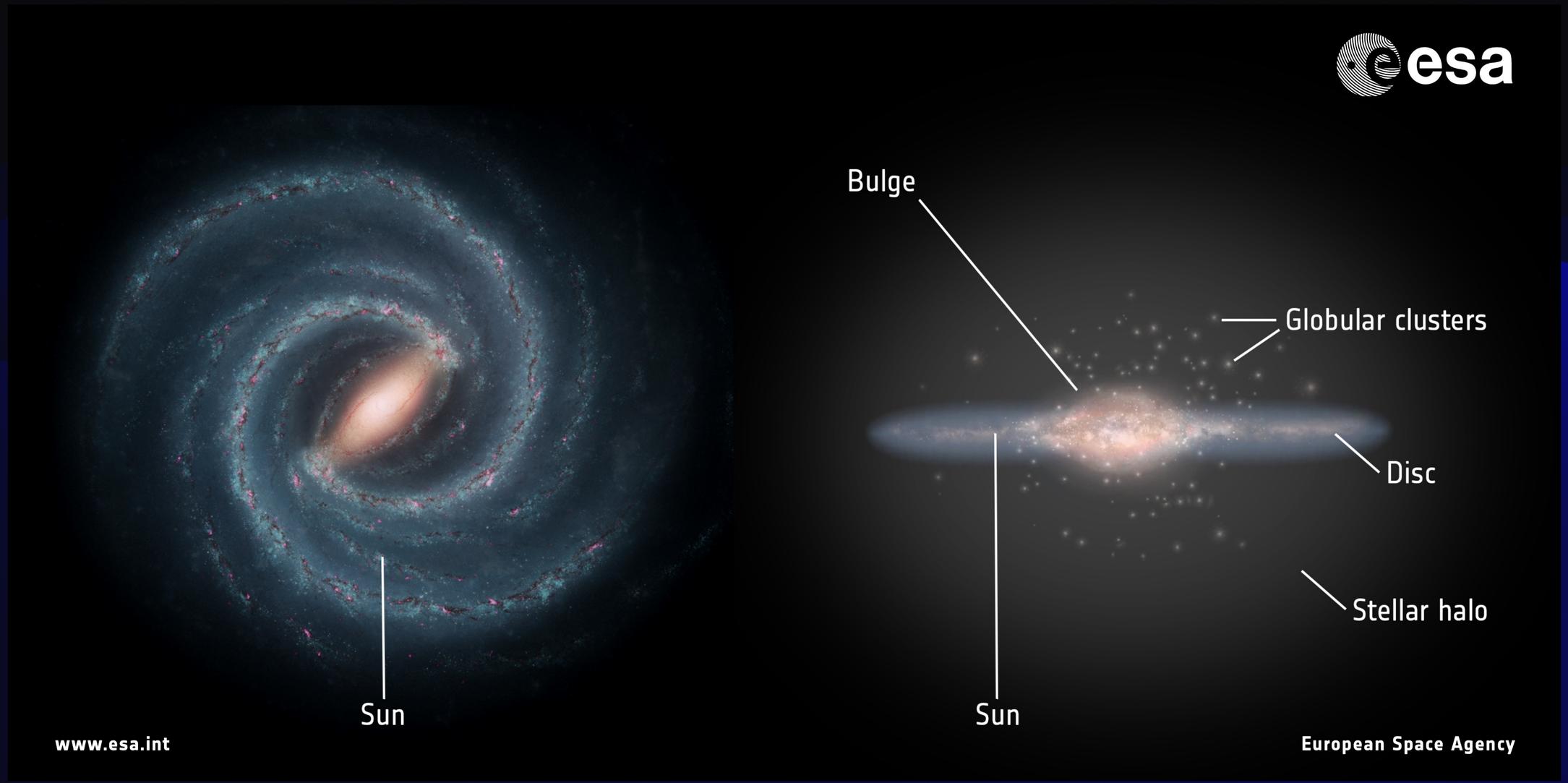
‘Exploring the Geometry, Topology and Morphology of Large Scale Structur using Minkowski Functionals’, Sheth et al. 2005.

$$\Lambda CDM : \frac{dn}{dM} \propto M^{-1.9}$$

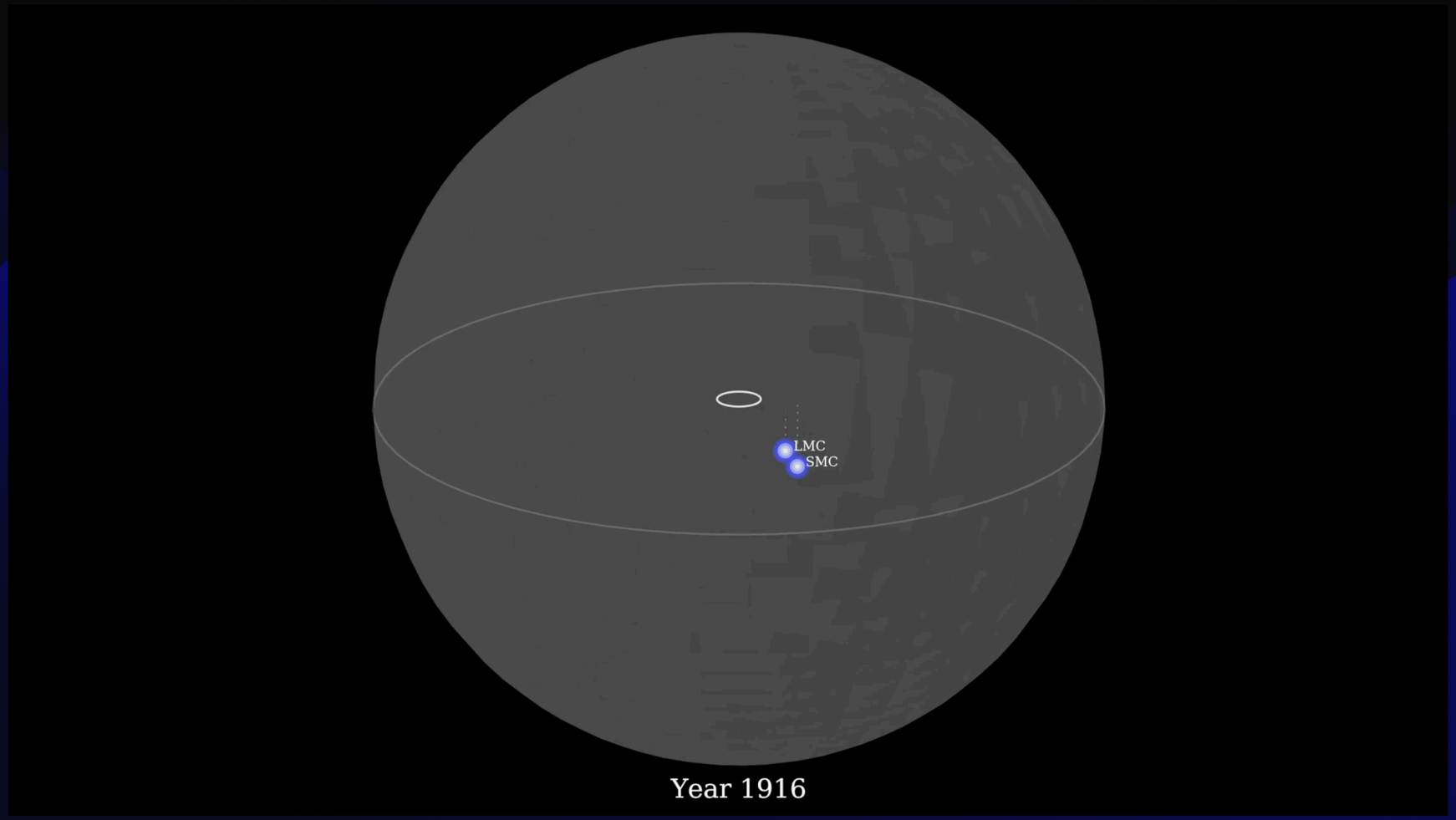


Stellar Halo

Anatomy of the Milky Way

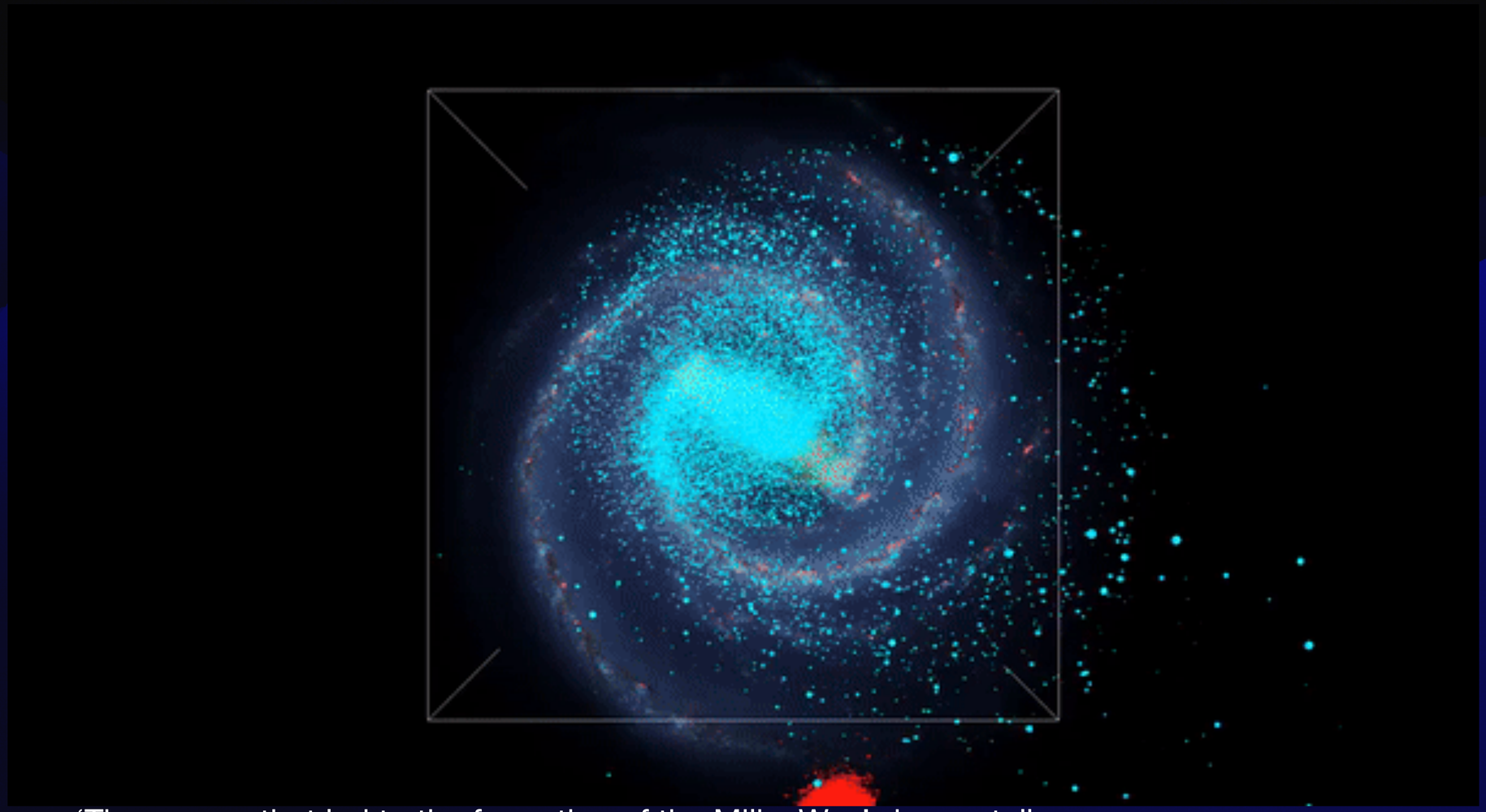


Milky Way satellite map



‘Small-Scale Challenges to the ΛCDM Paradigm’, Bulloc et al. 2017

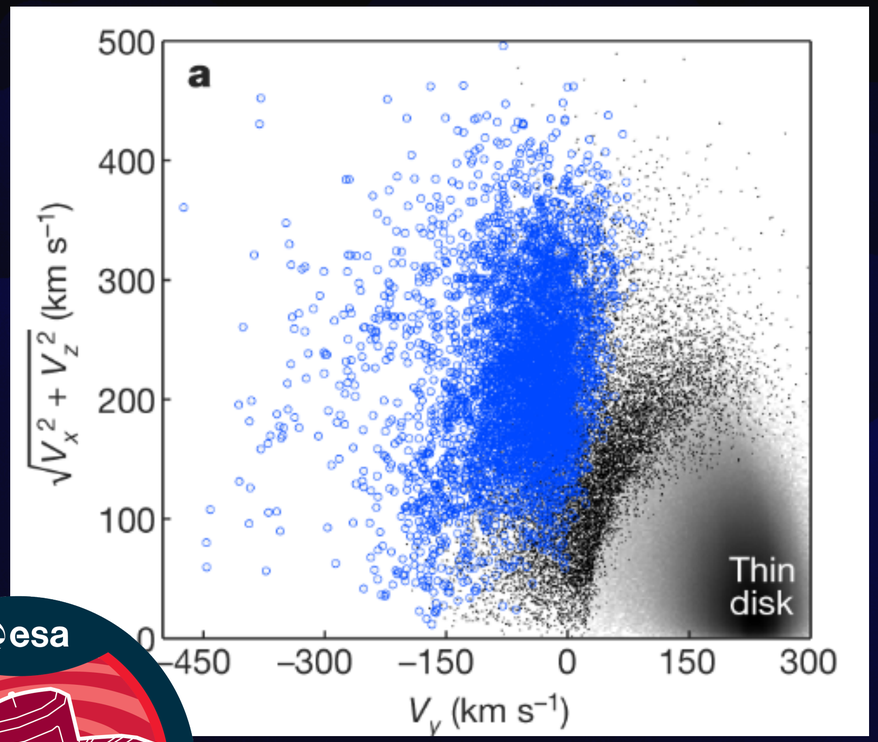
Merger simulation



‘The merger that led to the formation of the Milky Way’s inner stellar halo and thick disk’, Helmi et al. 2018

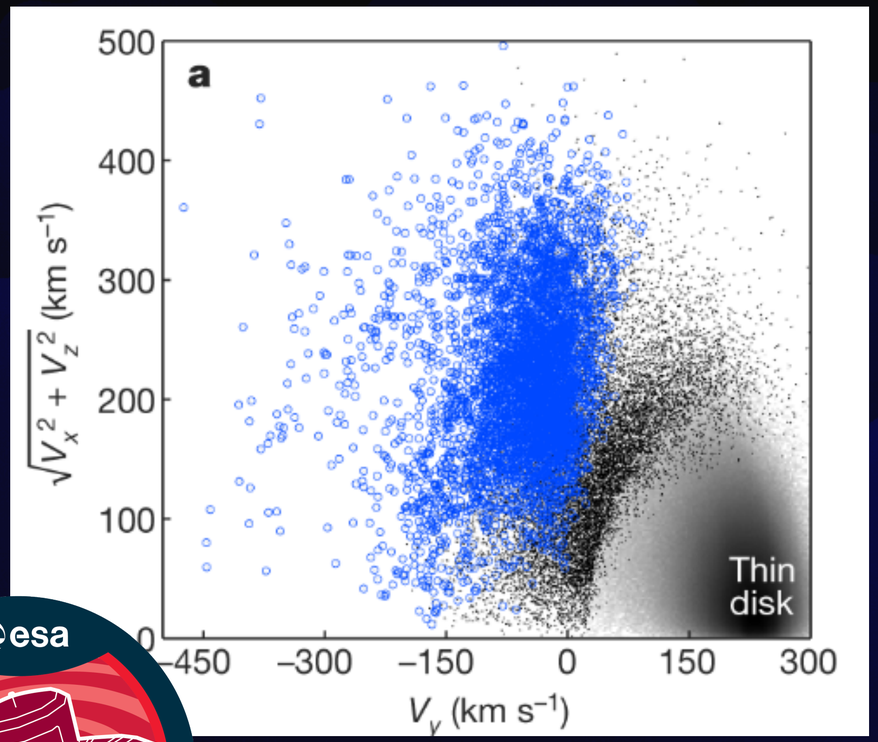
Galactic Archaeology

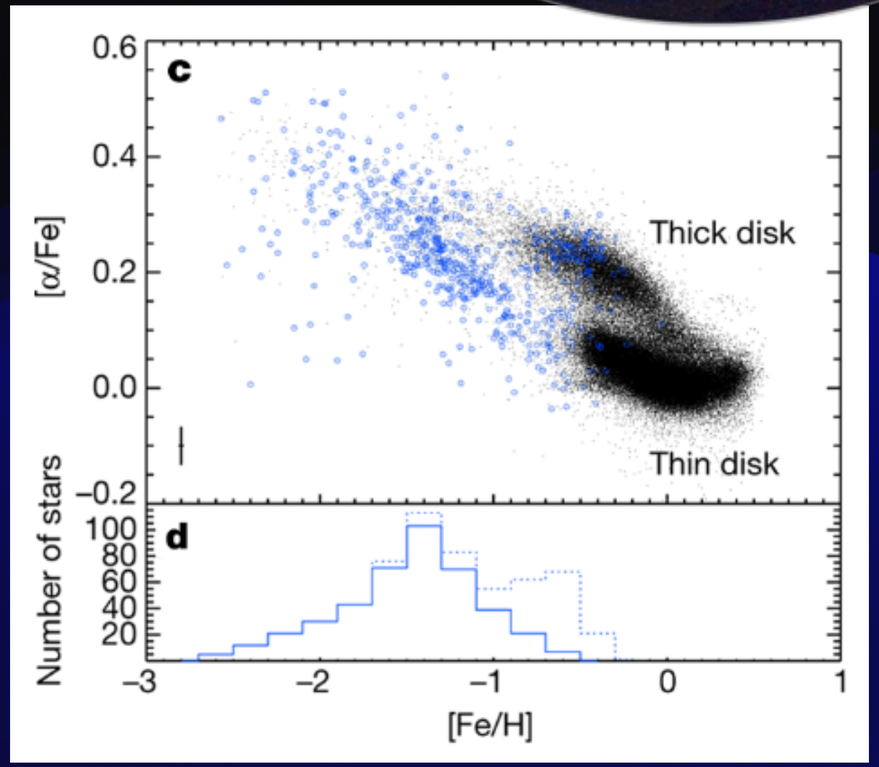
Dynamics: $(\vec{x}, \vec{v})$



‘The merger that led to the formation of the Milky Way’s inner stellar halo and thick disk’, Helmi et al. 2018

Dynamics: $(\vec{x}, \vec{v})$



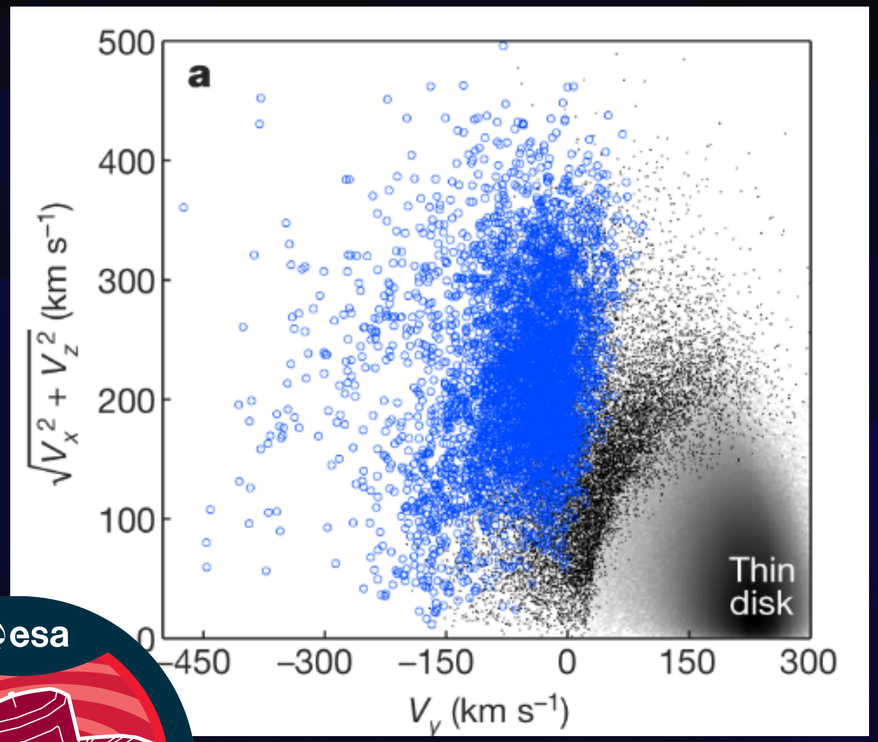


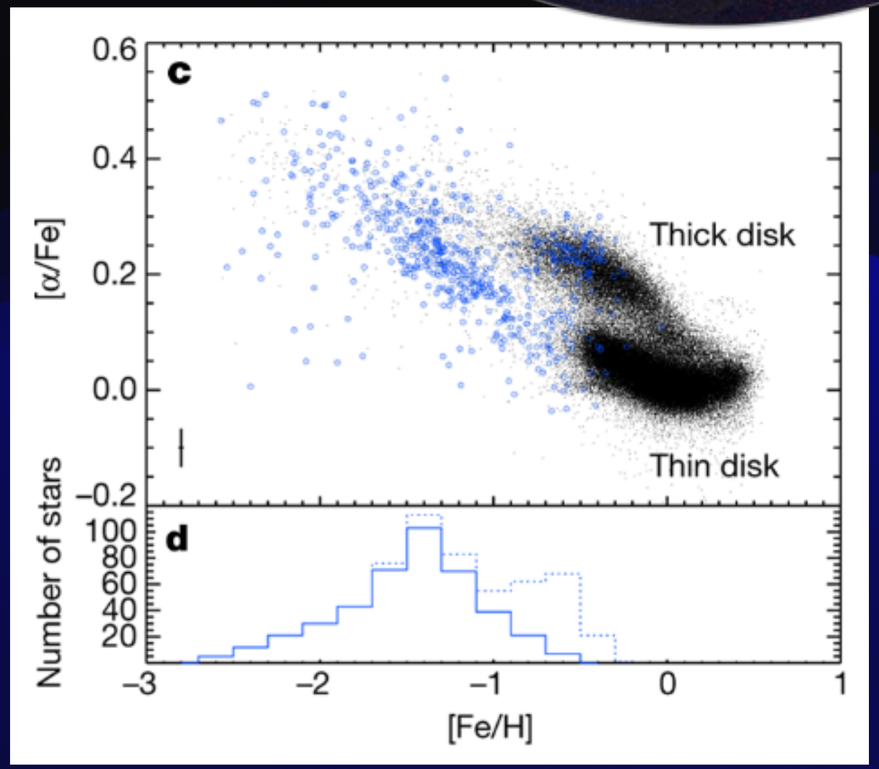
Chemistry: (Z, α)

‘The merger that led to the formation of the Milky Way’s inner stellar halo and thick disk’, Helmi et al. 2018

Gaia Sausage-Enceladus (GS/E)

Dynamics: $(\vec{x}, \vec{v})$



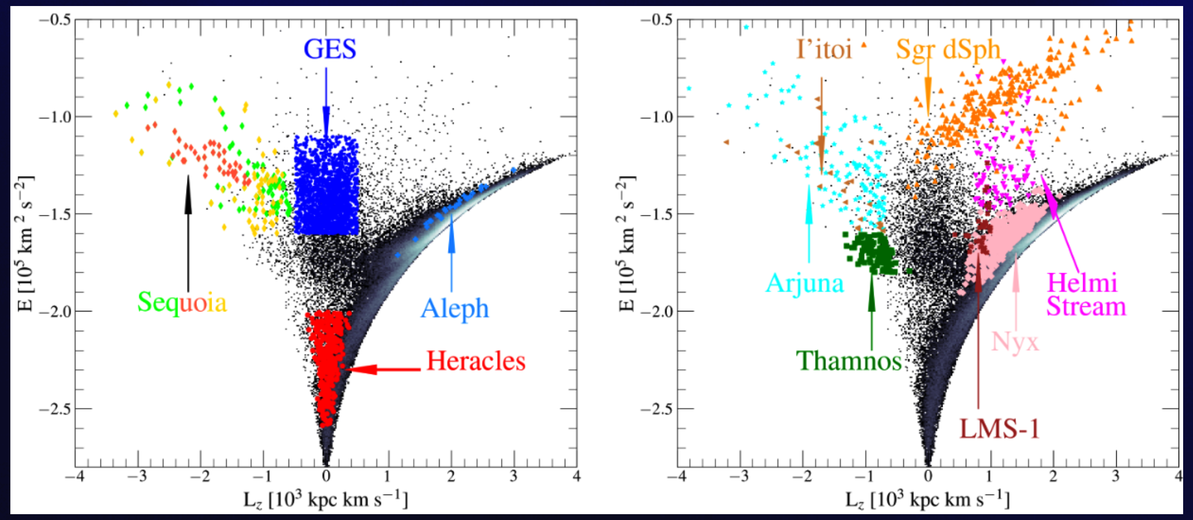


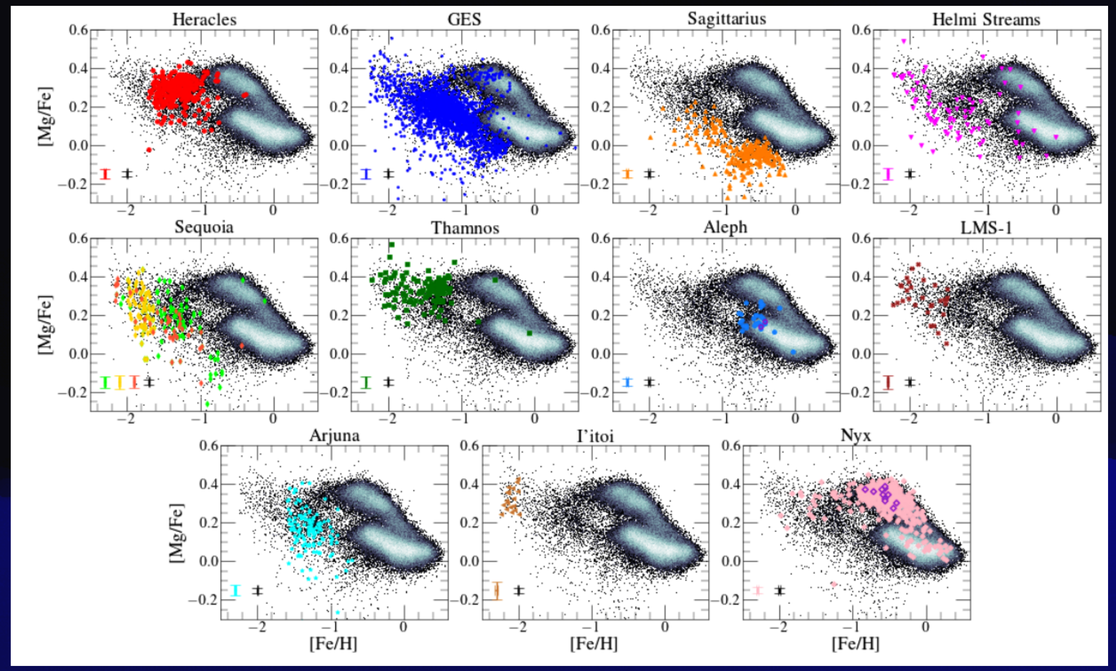
Chemistry: (Z, α)

‘The merger that led to the formation of the Milky Way’s inner stellar halo and thick disk’, Helmi et al. 2018

The modern picture

Dynamics



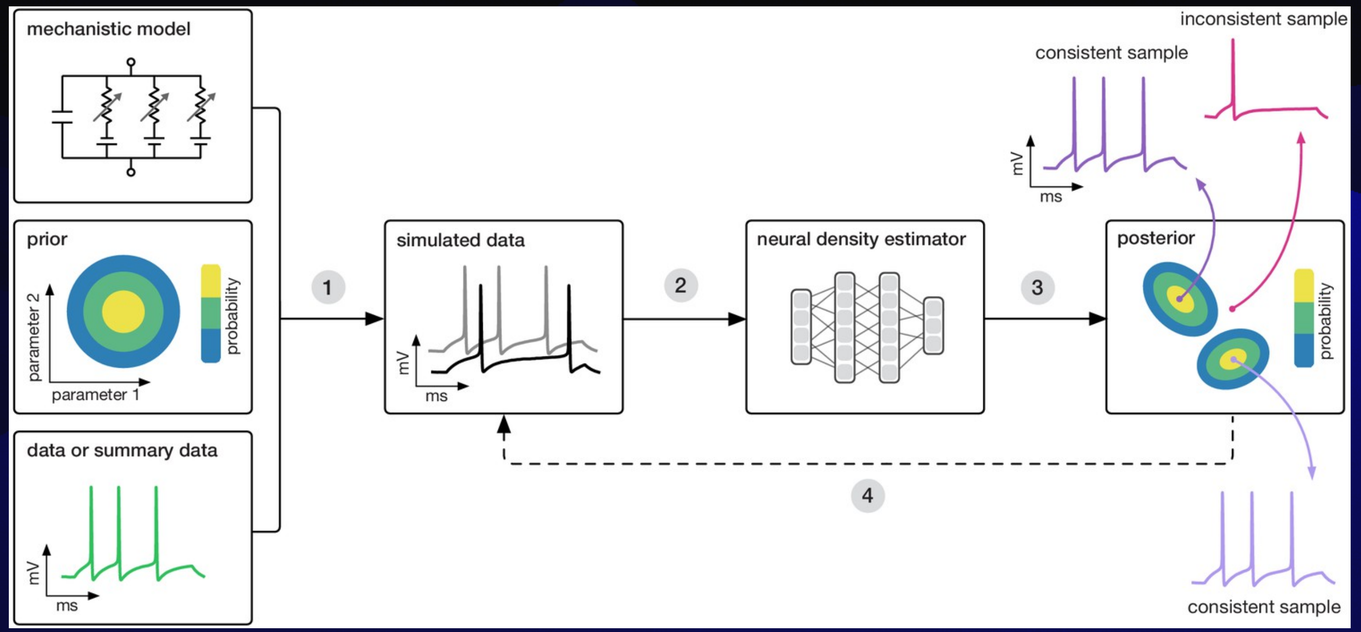


Chemistry

'On the mass assembly history of the Milky Way: clues from its stellar halo', Horta et al. 2024

Simulation Based Inference (SBI)

Train a Neural Density Estimate (NDE) of the Posterior $p(\theta \mid x)$ on (θ, x) pairs



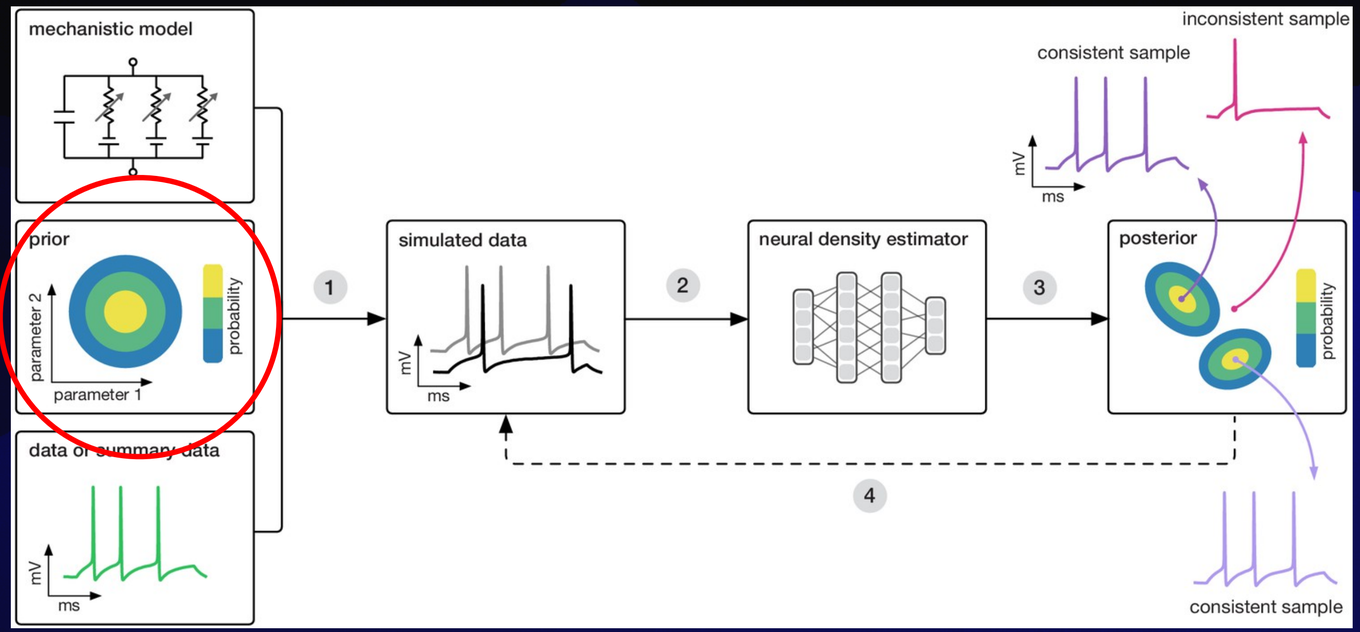
"Training deep neural density estimators to identify mechanistic models of neural dynamics“, Gonçalves, et al. 2024

Simulation Based Inference (SBI)

Train a Neural Density Estimate (NDE) of the Posterior $p(\theta \mid x)$ on (θ, x) pairs

Ingredients:

- Prior $p(\theta)$



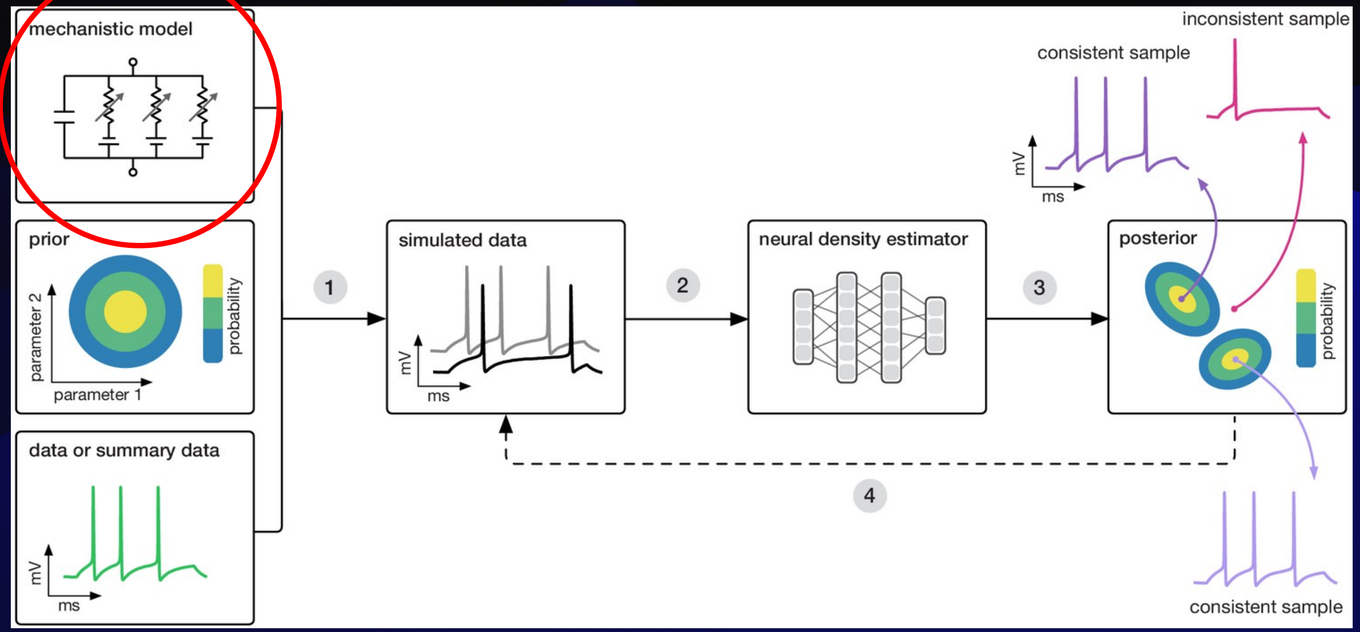
"Training deep neural density estimators to identify mechanistic models of neural dynamics“, Gonçalves, et al. 2024

Simulation Based Inference (SBI)

Train a Neural Density Estimate (NDE) of the Posterior $p(\theta \mid x)$ on (θ, x) pairs

Ingredients:

- Prior $p(\theta)$
- Simulator S



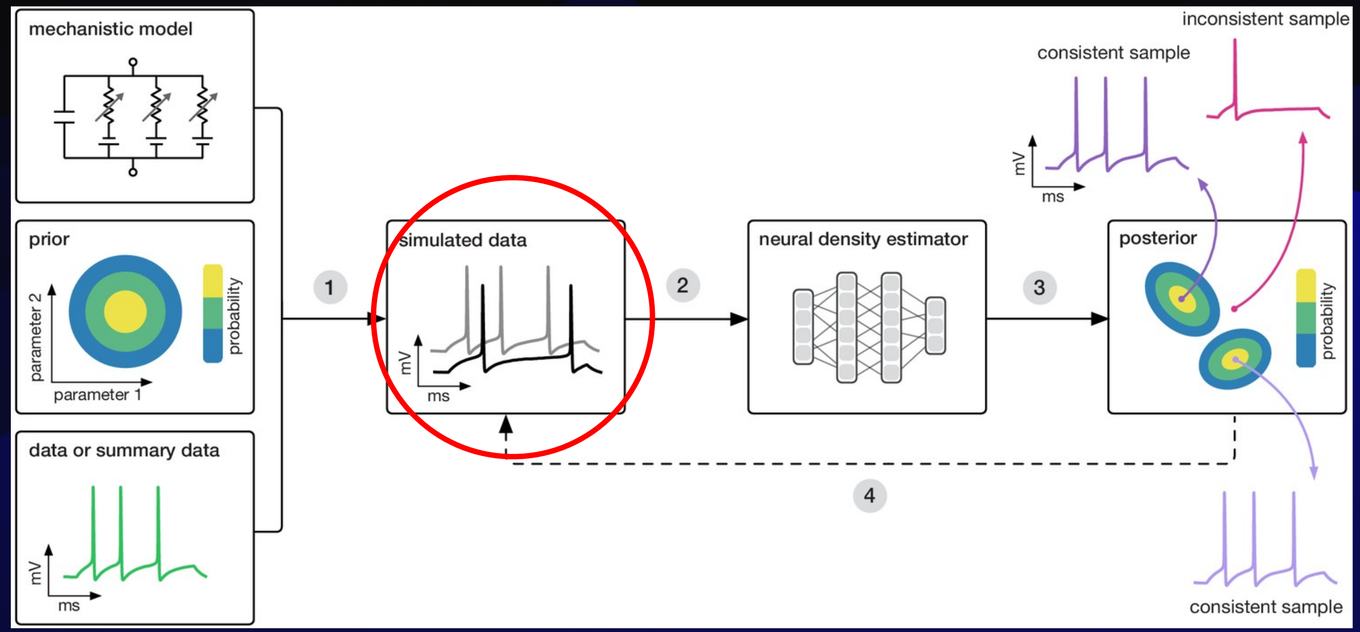
"Training deep neural density estimators to identify mechanistic models of neural dynamics“, Gonçalves, et al. 2024

Simulation Based Inference (SBI)

Train a Neural Density Estimate (NDE) of the Posterior $p(\theta \mid x)$ on (θ, x) pairs

Ingredients:

- Prior $p(\theta)$
- Simulator S
- Observations $x = S(\theta)$



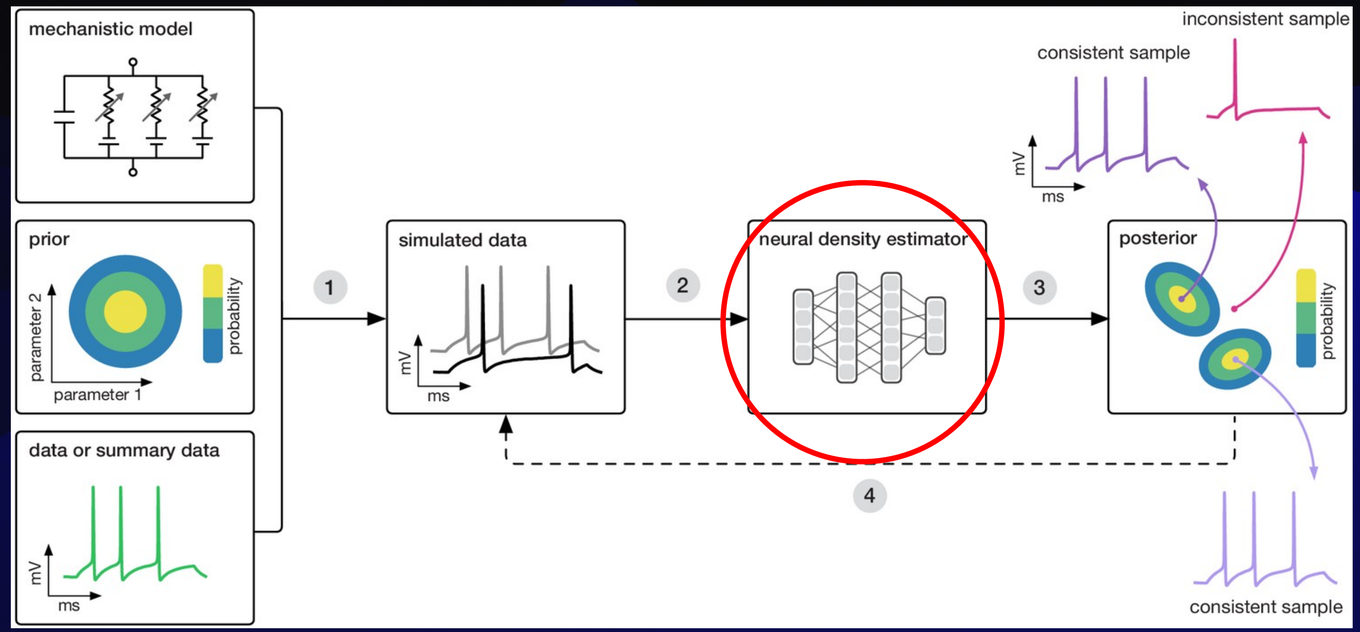
"Training deep neural density estimators to identify mechanistic models of neural dynamics“, Gonçalves, et al. 2024

Simulation Based Inference (SBI)

Train a Neural Density Estimate (NDE) of the Posterior $p(\theta \mid x)$ on (θ, x) pairs

Ingredients:

- Prior $p(\theta)$
- Simulator S
- Observations $x = S(\theta)$
- NDE $q_\phi(\theta \mid x) \sim p(\theta \mid x)$

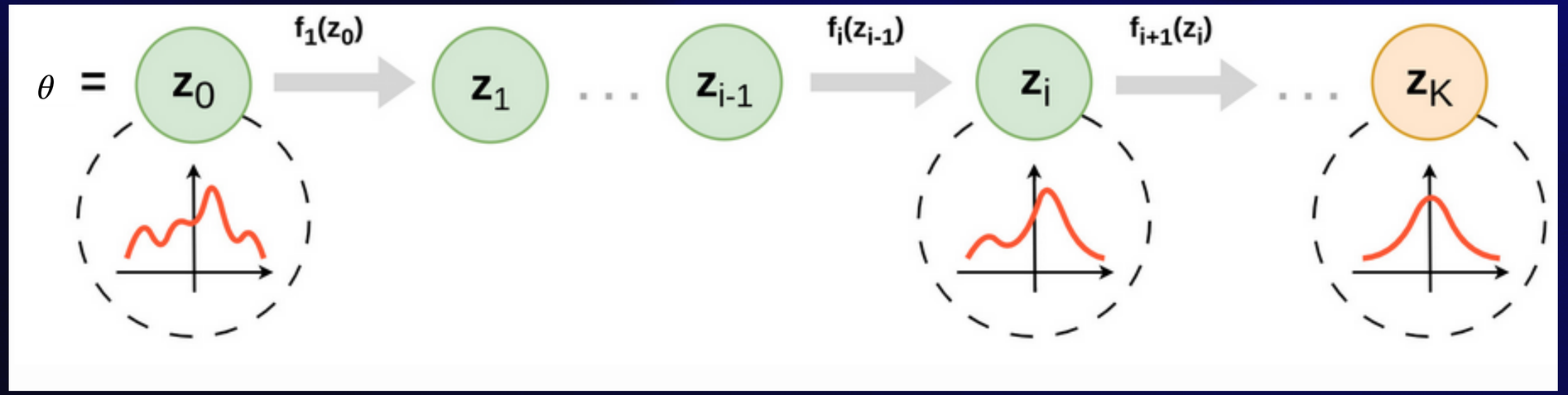


"Training deep neural density estimators to identify mechanistic models of neural dynamics“, Gonçalves, et al. 2024

Neural Density Estimation (NDE)

Normalizing Flow (NF):

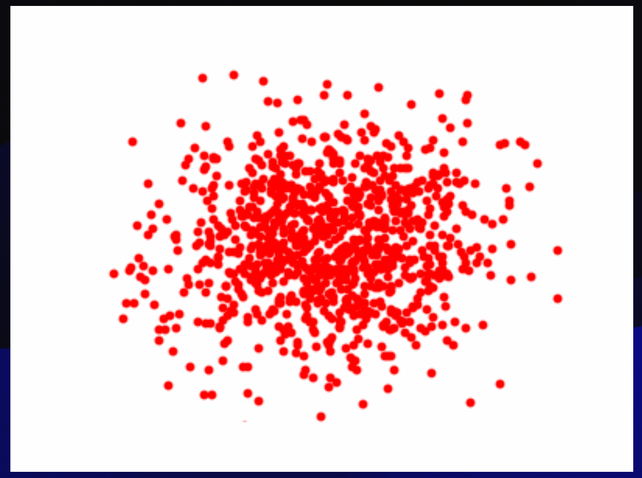
- $q_\phi(\theta \mid x) \sim p(\theta \mid x)$ -> conditional probability !
- Trained on target distribution samples

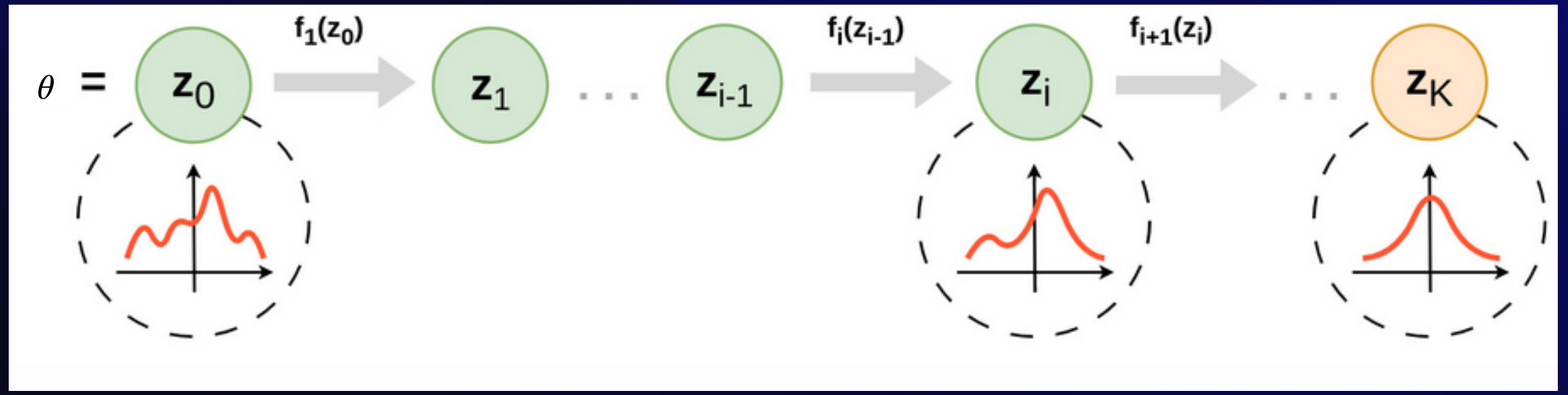


Neural Density Estimation (NDE)

Normalizing Flow (NF):

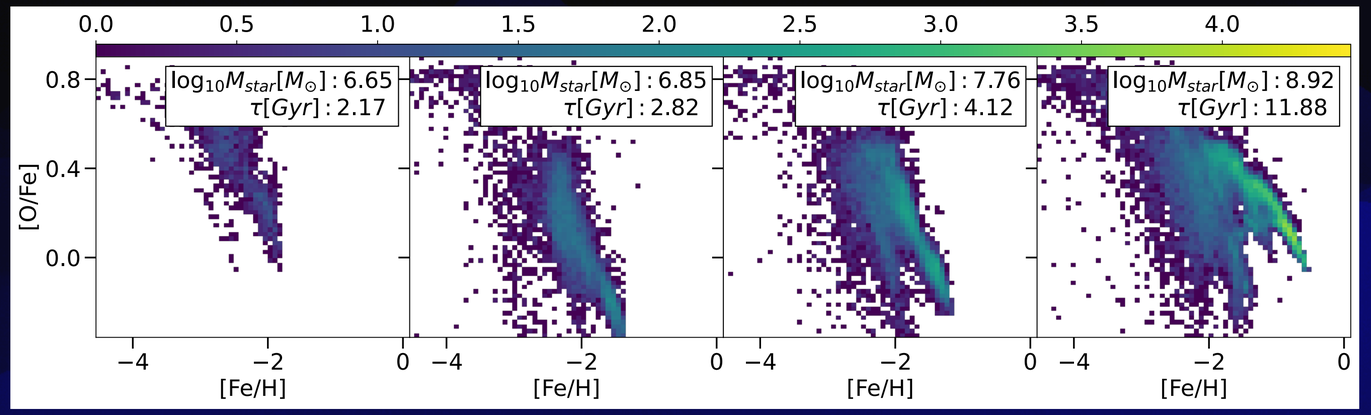
- $q_\phi(\theta|x) \sim p(\theta|x)$ -> conditional probability !
- Trained on target distribution samples





Galactic Archaeology SBI

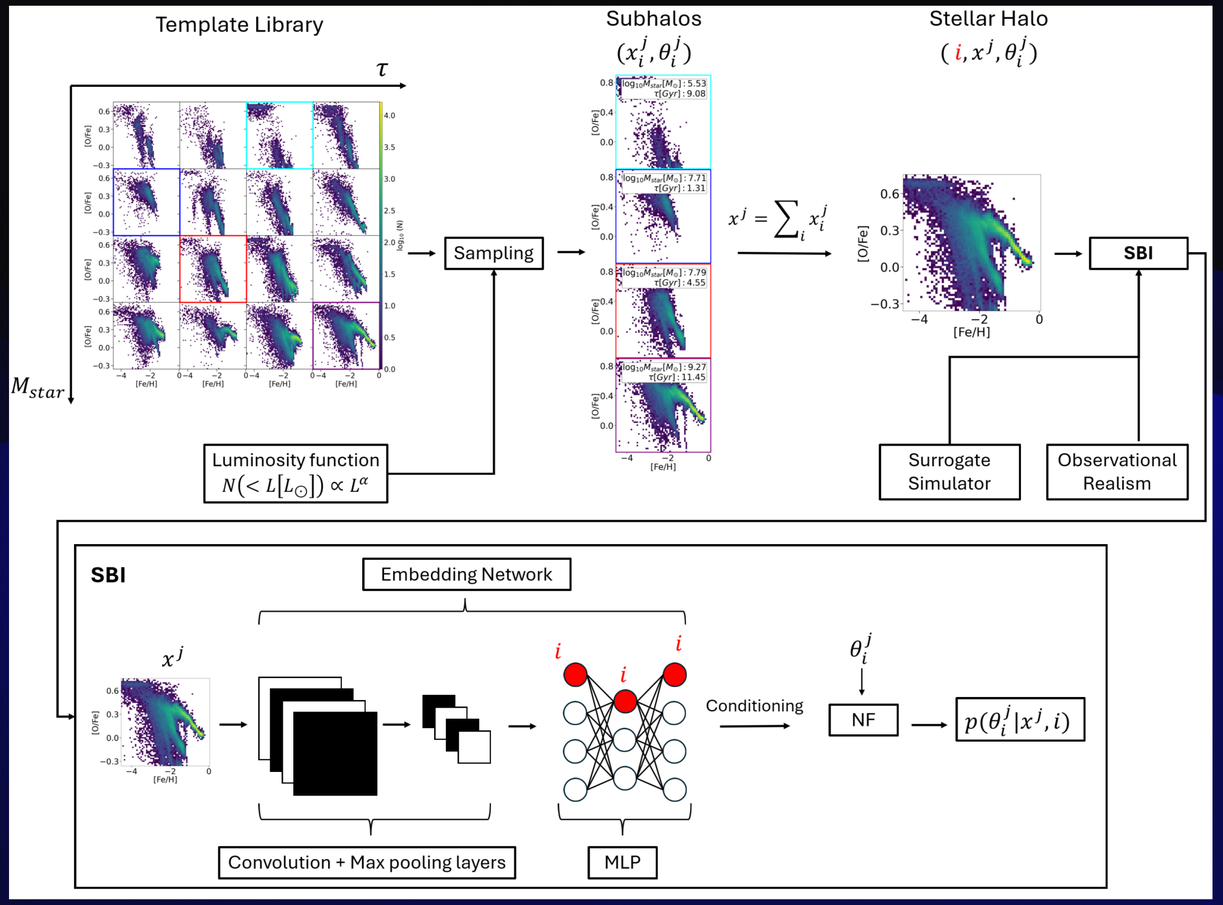
- With chemistry we constrained the stellar mass



CASBI: chemical abundance simulation based inference
https://arxiv.org/abs/2411.17269

Galactic Archaeology SBI

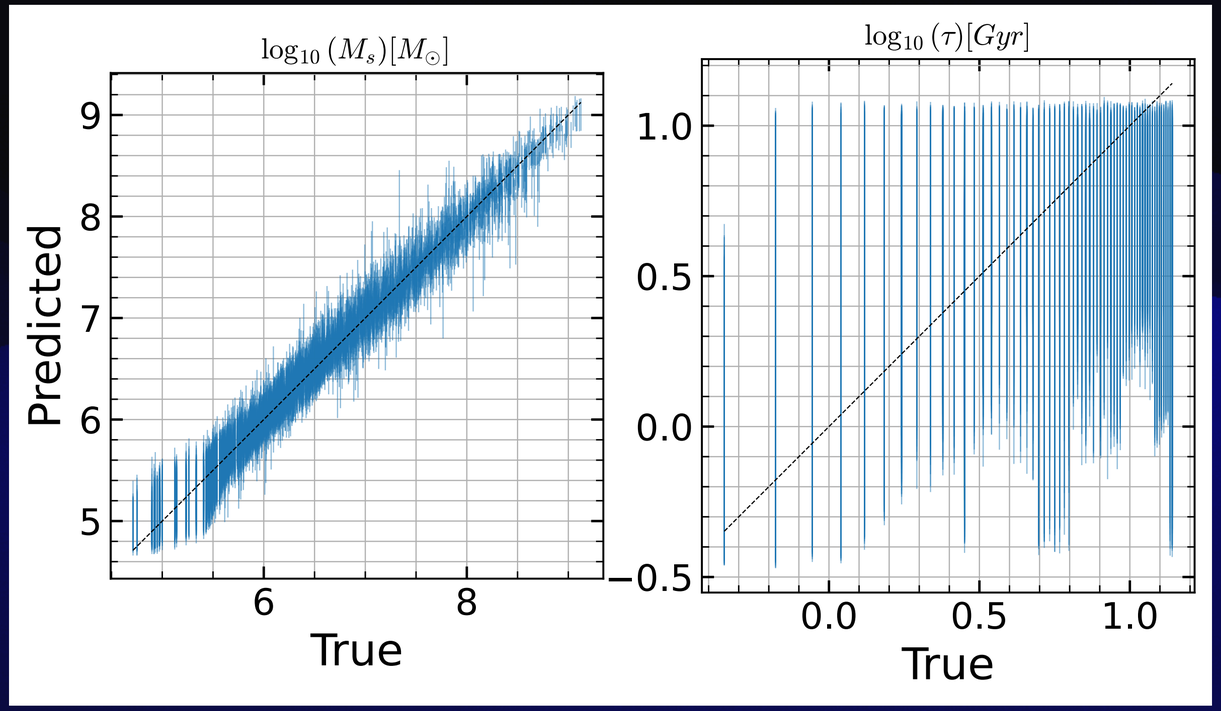
- With chemistry we constrained the stellar mass



CASBI: chemical abundance simulation based inference
https://arxiv.org/abs/2411.17269

Galactic Archaeology SBI

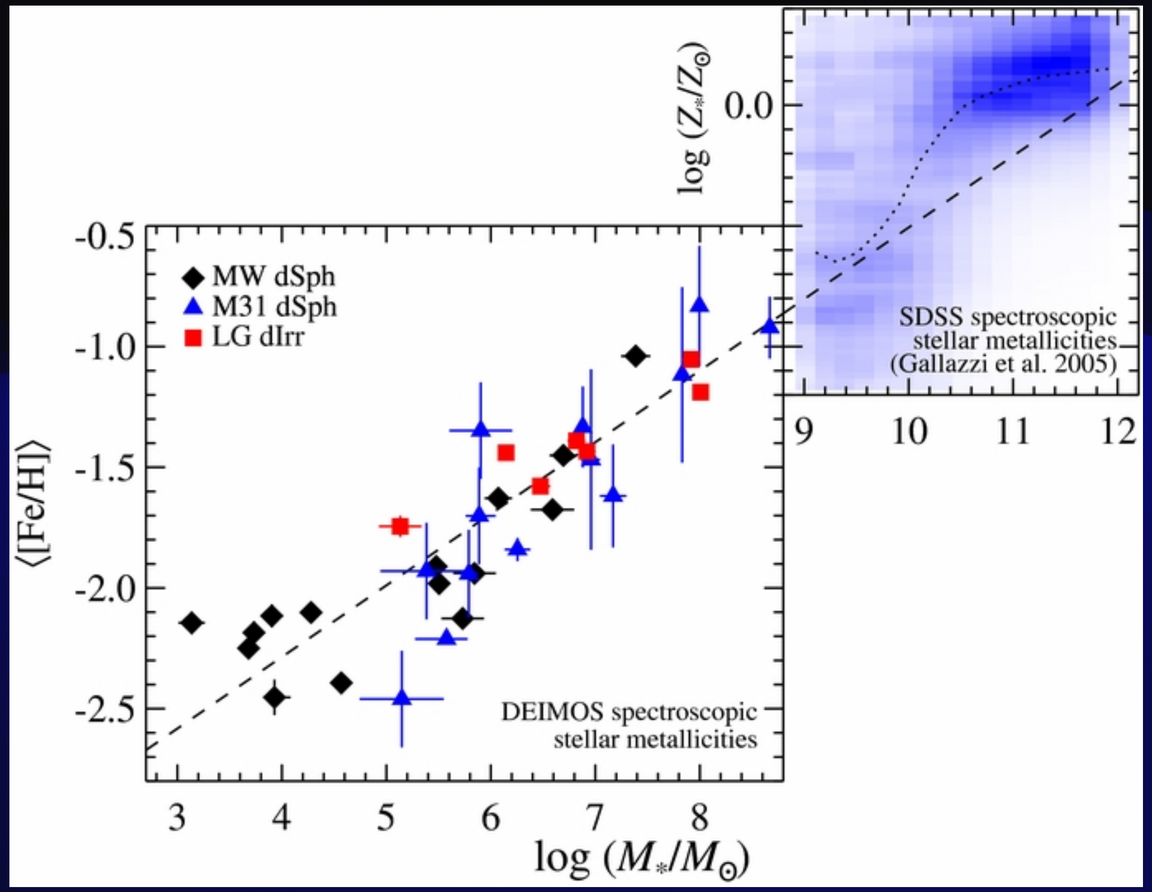
- With chemistry we constrained the stellar mass
- Bad results on the infall time



CASBI: chemical abundance simulation based inference
https://arxiv.org/abs/2411.17269

Galactic Archaeology SBI

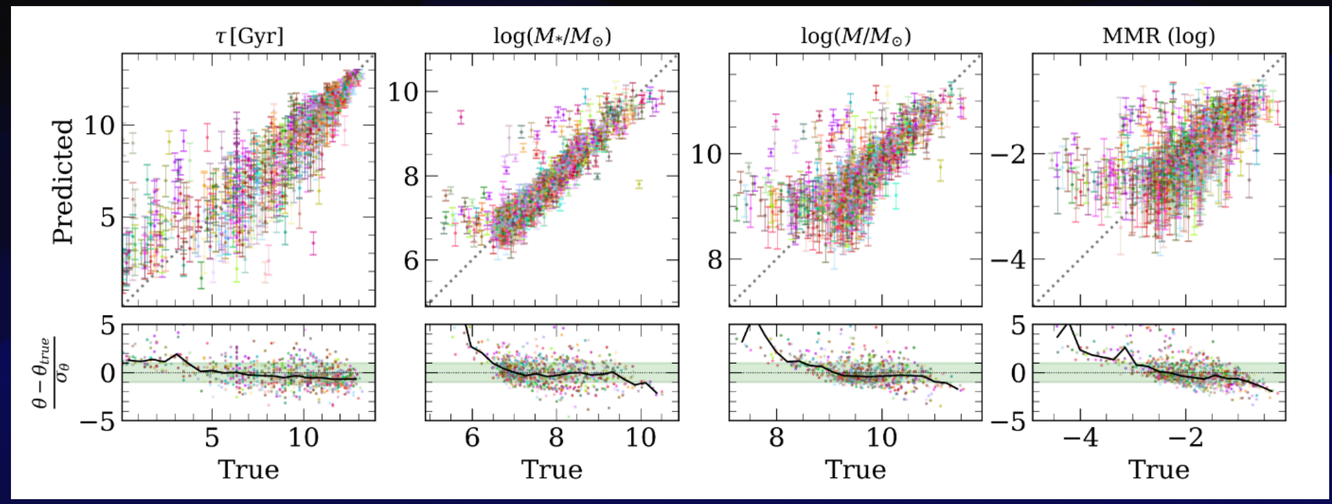
- With chemistry we constrained the stellar mass
- Bad results on the infall time



'The universal stellar mass-stellar metallicity relation for dwarf galaxies', Kirby et al. 2013

Galactic Archaeology SBI

- With chemistry we constrained the stellar mass
- Bad results on the infall time
- Chemistry + Dynamics have (recently) proven to be good observables for GA-SBI



"GalactiKit: reconstructing mergers from $z = 0$ debris using simulation-based inference in Auriga", Sante et al. 2025