Hybrid Physics-AI Approach for Cloud Cover Nowcasting

R. Elmontassir[1], O. Pannekoucke[1,2], C. Lapeyre[1]
[1]CERFACS, Toulouse, France
[2]INPT-ENM, CNRM UMR 3589, Météo-France/CNRS, Toulouse, France

1. Context

Recent works using AI have explored cloud cover nowcasting, showing that a future sequence of satellite images can be predicted from a sequence of past observed images, but with some defects: it is not always possible to transport structures that are abnormally attenuated. Moreover, using numerical resources (data, computation) to learn physical processes that are already known, e.g. the advection, to the detriment of more complex processes, appears to be a waste of resources. To tackle these issues, we propose a novel hybridization between physics and AI, where part of the known physical equations is mixed with state-of-the-art Deep Learning techniques.

In this work we designed a hybrid Physics-AI model that enforces a physical behaviour, transport dynamics, as a hard constraint for a trained U-Net architecture

2. PDE-NetGen

The goal of this work is to propose an architecture bridging AI and physics, applied to cloud cover nowcasting. To meet this objective, we used PDE-NetGen[1].

PDE-NetGen

A package providing tools to automatically translate physical equations, given as PDEs of this type :

$$\partial_t f = F\left(f, \partial f, .., \partial^\alpha f\right) \tag{1}$$

into neural networks.

For temporal integration, a numerical scheme can be used directly (Euler, RK4,. . .) or in the form of a network. Thus expressed in the form of a network, the physical part can be combined with other neural networks to be calibrated, the whole forming a network which can be trained using available conventional tools (here Keras).

$$\frac{\partial^2 h}{\partial x^2}(x_i) \approx \frac{h_{i+1} - 2h_i + h_{i-1}}{\delta x^2}$$

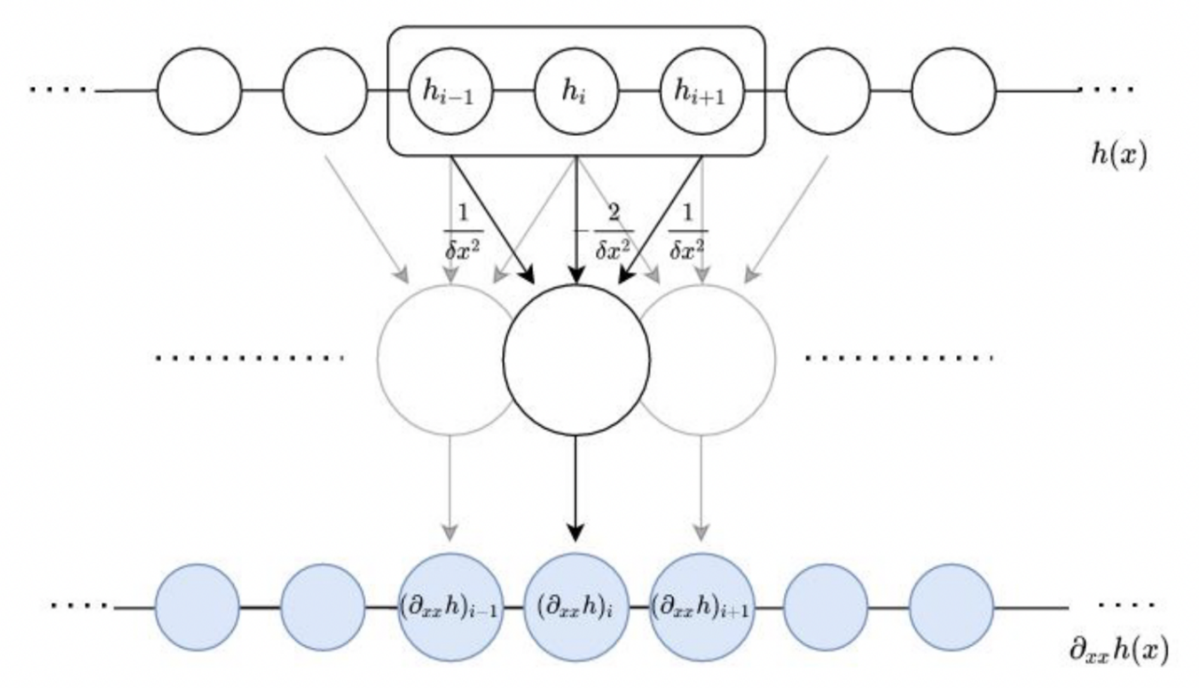
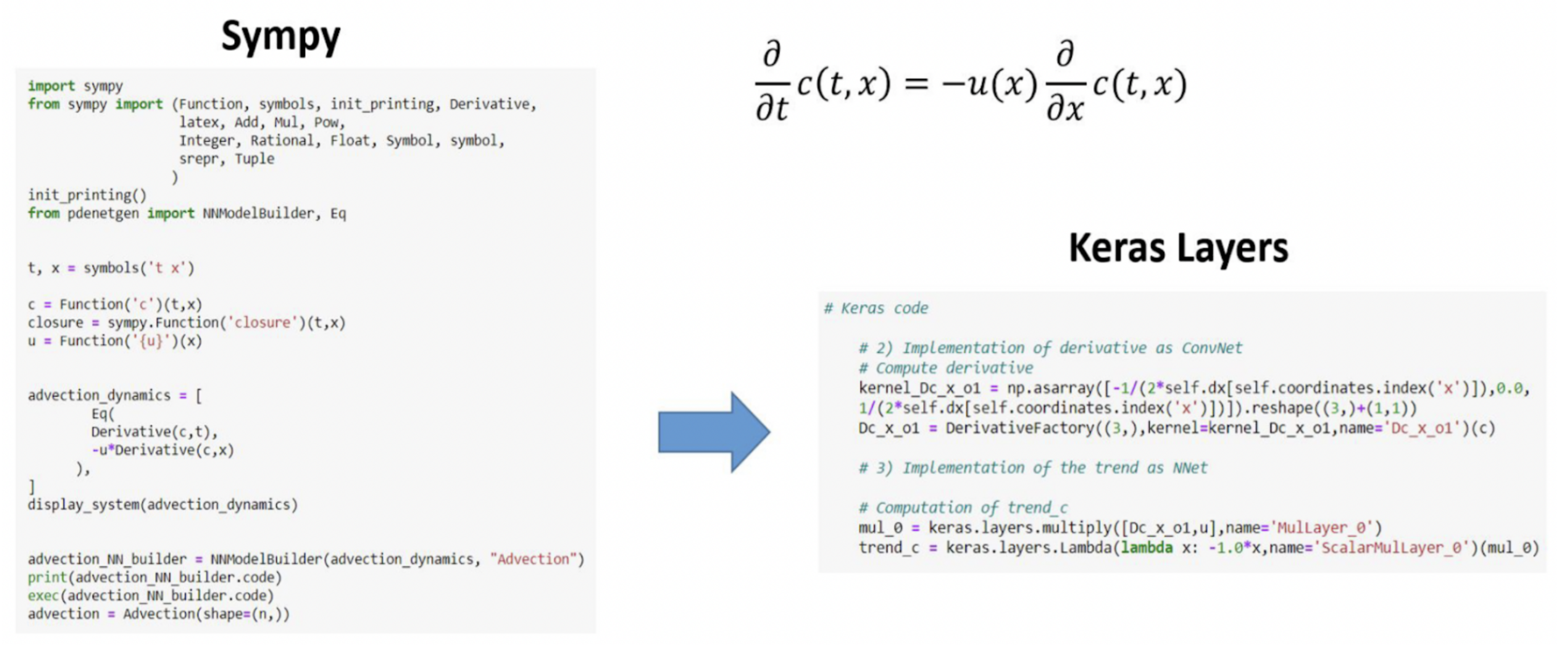


Figure: The link between finite differences and convolutional layers.



Sympy

```
import sympy
from sympy import (Function, symbols, init_printing, Derivative,
                   latex, Add, Mul, Pow,
                   Integer, Rational, Float, Symbol, symbol,
                   srepr, Tuple
                  )
init_printing()
from pdenetgen import NNModelBuilder, Eq


t, x = symbols('t x')

c = Function('c')(t,x)
closure = sympy.Function('closure')(t,x)
u = Function('{u}')(x)


advection_dynamics = [
        Eq(
        Derivative(c,t),
        -u*Derivative(c,x)
       ),
]
display_system(advection_dynamics)


advection_NN_builder = NNModelBuilder(advection_dynamics, "Advection")
print(advection_NN_builder.code)
exec(advection_NN_builder.code)
advection = Advection(shape=(n,))
```

$$\frac{\partial}{\partial t}c(t,x) = -u(x)\frac{\partial}{\partial x}c(t,x)$$

Keras Layers

```
# Keras code

    # 2) Implementation of derivative as ConvNet
    # Compute derivative
    kernel_Dc_x_o1 = np.asarray([-1/(2*self.dx[self.coordinates.index('x')]),0.0,
    1/(2*self.dx[self.coordinates.index('x')])]).reshape((3,)+(1,1))
    Dc_x_o1 = DerivativeFactory((3,),kernel=kernel_Dc_x_o1,name='Dc_x_o1')(c)

    # 3) Implementation of the trend as NNet

    # Computation of trend_c
    mul_0 = keras.layers.multiply([Dc_x_o1,u],name='MulLayer_0')
    trend_c = keras.layers.Lambda(lambda x: -1.0*x,name='ScalarMulLayer_0')(mul_0)
```

Figure: On the left the PDE written with Sympy and on the right the generated code.

3. The physical model

We used the transport equation as physical information to represent the transport of cloud by winds:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = 0 \tag{2}$$

With u the velocity field and f the pixel values of the image.

1. First part : Generated by PDE-NetGen which translates the generator F of the equation 1, where the derivatives are transformed into convolutional layers with fixed kernels coming from the formulation in finite differences using a Python computer algebra library called Sympy. This neural network takes as input the initial condition (state at $t = 0$) and velocity (U-Net's output).
2. Second part : A Runge–Kutta fourth-order numerical scheme implemented as a neural network, used to solve the PDE.

4. The Hybrid model

Our idea is to put a U-Net [3] architecture upstream of the physical model. This U-Net takes as input a sequence of images, mainly, the state at t_{-1} and at t_{-2}. In output, we get a field, $u = (u, v)$ which will in turn be given as input to the physical model. And during training, the physical model will push the U-Net to the corresponding velocity fields.

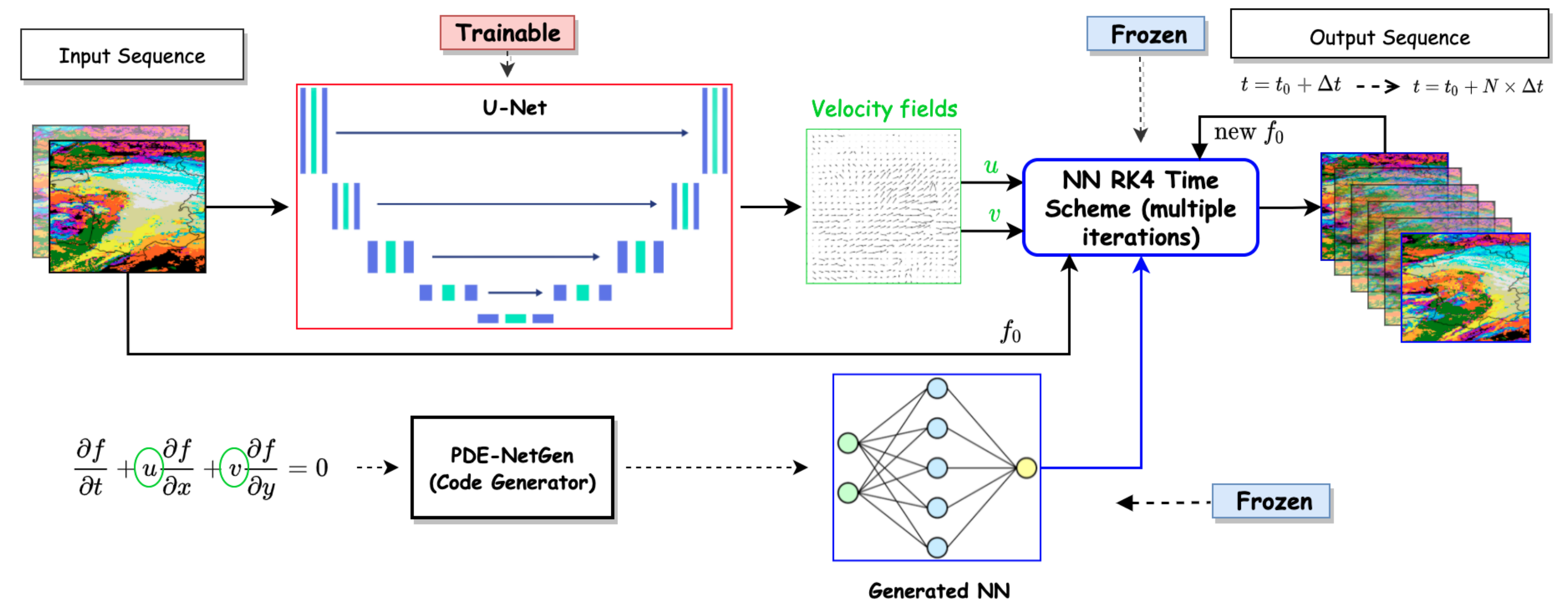

Input Sequence
Trainable
U-Net
Velocity fields
u
v
Frozen
NN RK4 Time Scheme (multiple iterations)
Output Sequence
$t = t_0 + \Delta t$ - -> $t = t_0 + N \times \Delta t$
new f_0
f_0
$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = 0$
PDE-NetGen (Code Generator)
Generated NN
Frozen

Figure: The hybrid model - named **NebulRK4** -

5. Results

We trained this model over 3 years of satellite images with Cloud Type classification.

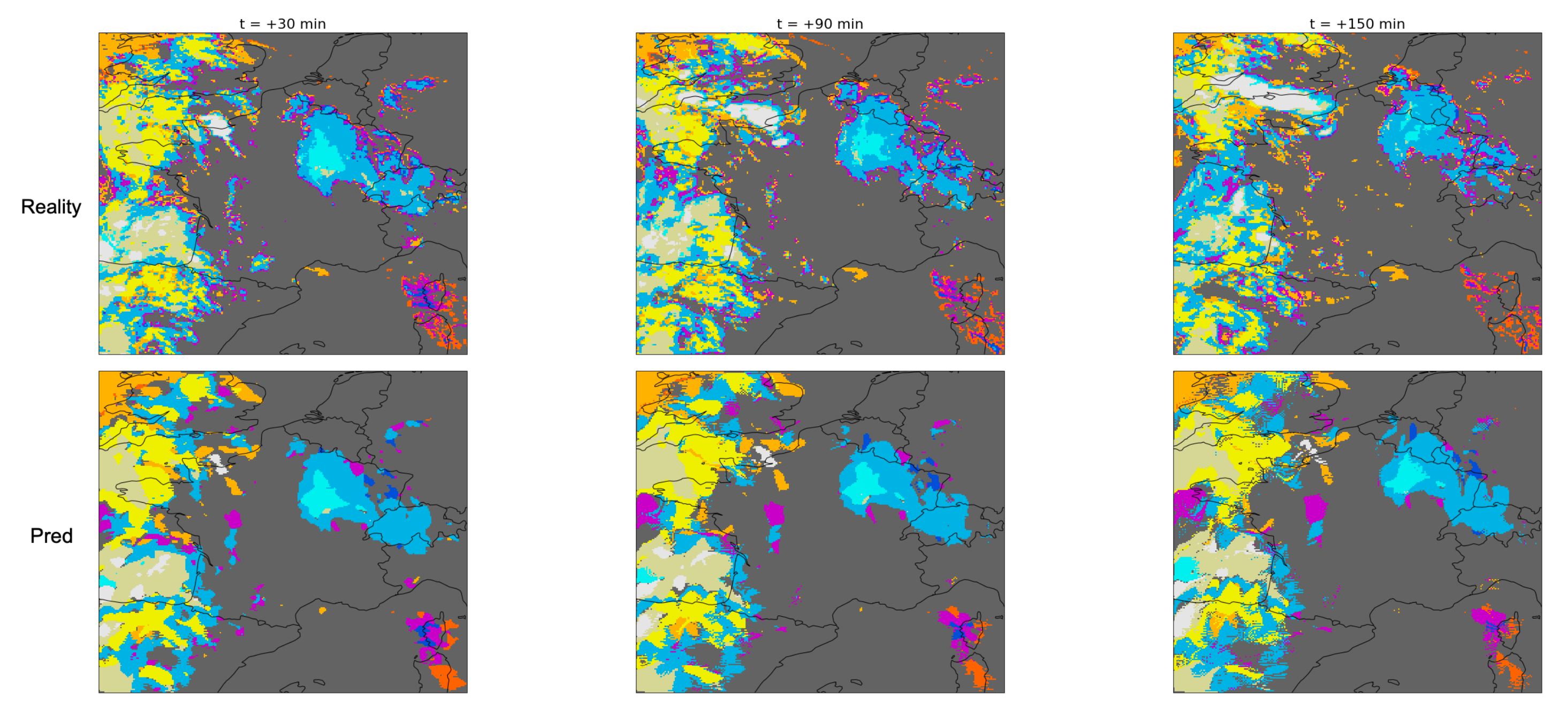


Figure: A single prediction at T + 30, T + 90 and T + 150 min lead time

5. Results

✓ This learning is indirectly supervised.
✓ The U-Net manages to estimate a consistent velocity.
✓ The cloud structure is preserved during the forecasting steps.

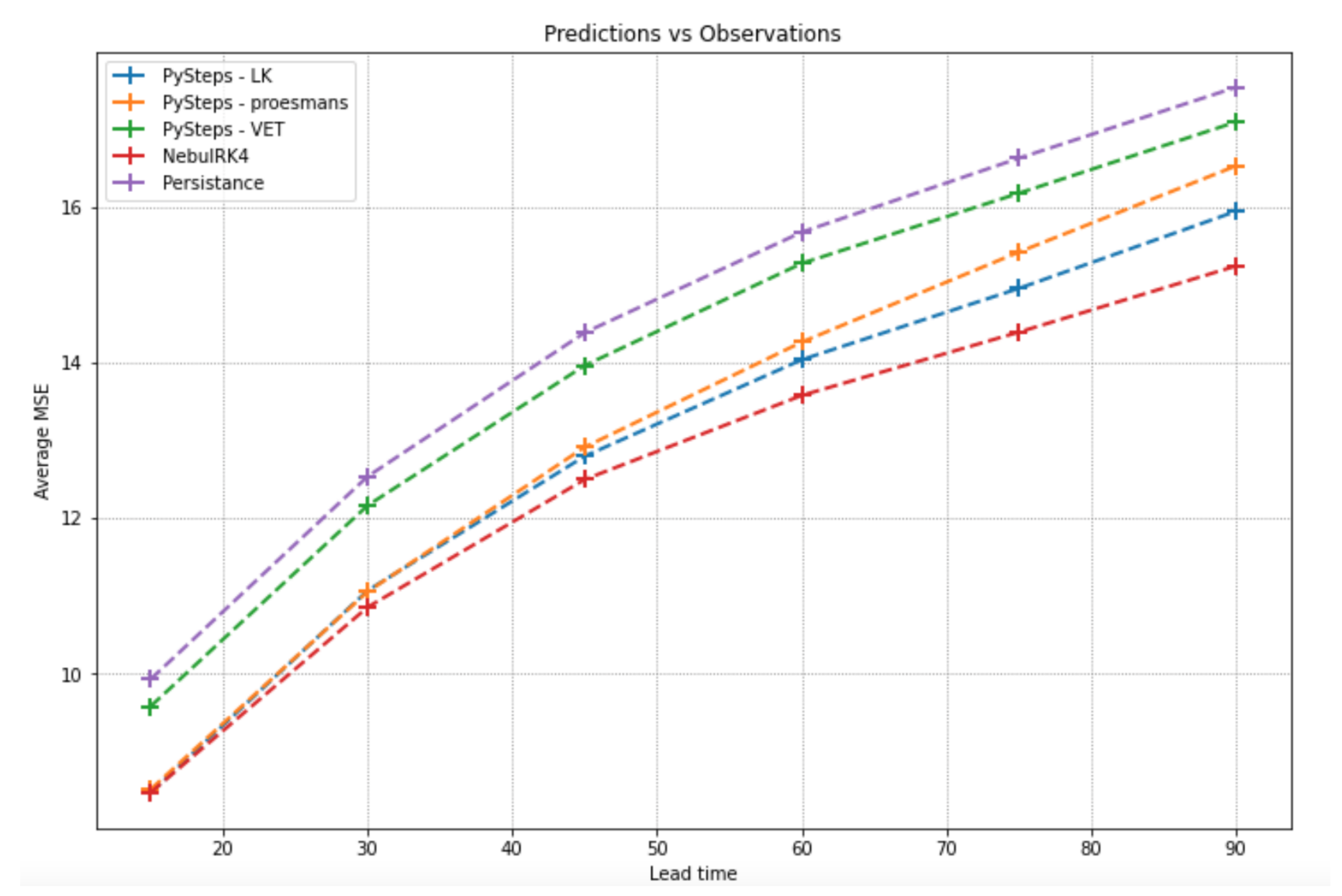


Figure: MSE score comparison between Persistance, NebulRK4 and 3 optical flow methods implemented on PySteps[2] (Lucas-Kanade (LK), Anisotropic diffusion method (Proesmans) and Variational echo tracking (VET))

✓ NebulRK4 outperforms Persistance.
✓ NebulRK4 outperforms Optical flow methods implemented on PySteps on the long term.

6. Conclusions and Perspectives

✓ Building hybrid architecture taking into account PDEs as physical information.
✓ Establishing a bridge between neural networks (statistical approach) and physics (equation of nature) to capitalize on the skills and knowledge of these two approaches.
✓ Easy-to-handle architecture.
✓ Integrating automatic code generation.
✓ Stability constraints (e.g., the CFL) have to be verified for the training to succeed.

7. References

Olivier Pannekoucke and Ronan Fablet.
Pde-netgen 1.0: from symbolic partial differential equation (pde) representations of physical processes to trainable neural network representations.
Geoscientific Model Development, 13(7):3373–3382, 2020.

Seppo Pulkkinen, Daniele Nerini, Andrés Pérez Hortal, Carlos Velasco-Forero, Alan Seed, Urs Germann, and Loris Foresti.
Pysteps: an open-source python library for probabilistic precipitation nowcasting (v1.0).
Geoscientific Model Development, 12:4185–4219, 10 2019.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
U-net: Convolutional networks for biomedical image segmentation, 2015.