Ph. Bykov,
Hydrometeorological Research Center of the Russian Federation

Machine Learning for postprocessing at RHM

Gradient Machine Learning (ML)

Let's consider the some piecewise-smooth parametric tensor algorithm $\left\{\vec{Y}\right\} = F\left(\left\{\vec{X}\right\},\left\{\vec{\theta}\right\}\right)$, where $\left\{\vec{Y}\right\},\left\{\vec{X}\right\}$ and $\left\{\vec{\theta}\right\}$ – output, input and parameters sets of tensors (Nd arrays) respectively and the some $e\left(\left\{\vec{Y}\right\},\left\{\vec{Y}_{fact}\right\}\right)$ – the piecewise-smooth *loss function*. The ML minimize the mean value of loss function: $\sum_j e\left(F\left(\left\{\vec{X}\right\}_j,\left\{\vec{\theta}\right\}\right),\left\{\vec{Y}_{fact}\right\}_j\right) \to \min_{\left\{\vec{\theta}\right\}}$, where j – realization of tensors.

Gradient ML use the gradient descent (GD) method for minimization:

$$\left\{\vec{\theta}\right\}_{s+1} = \left\{\vec{\theta}\right\}_s - \eta_s \nabla_{\left\{\vec{\theta}\right\}} F\left(\left\{\vec{X}\right\},\left\{\vec{\theta}\right\}\right) \otimes \nabla_{\left\{\vec{Y}\right\}} e\left(\left\{\vec{Y}\right\},\left\{\vec{Y}_{fact}\right\}\right),\ \eta_s > 0. \qquad (1)$$

Examples: $F\left(\vec{X},\vec{\theta}\right) = \vec{\theta} \otimes \vec{X}$ – linear. If $e = \left\|\vec{Y} - \vec{Y}_{fact}\right\|_{L_2} \Rightarrow$ the least squares method $\Rightarrow$ no need GD. If $e = \left\|\vec{Y} - \vec{Y}_{fact}\right\|_{L_1} \Rightarrow$ the least absolute deviations $\Rightarrow$ need GD or other iterative method for minimization.

Deep Learning (DL)

DL is the subdomain of GML and considering the algorithms, compiled from operators from a special library (inheritors of some class). Those operators (classes) have forward method $\left\{\vec{X}_k\right\},\left\{\vec{\theta}_k\right\} \to F_k\left(\left\{\vec{X}_k\right\},\left\{\vec{\theta}_k\right\}\right)$ and backward propagation of the error method:

$$\nabla_e F_k \to \left\{\nabla_e \vec{X}_k\right\},\left\{\nabla_e \vec{\theta}_k\right\} \qquad (2)$$

DL can optimize parameters of any algorithm, compiled from such operators. The DL optimization is memory intensive: for apply (2) we need intermediate values $\left\{\vec{X}_k\right\}$. The most popular open source DL libraries:

- **Tensorflow** (by Google, 2.5M lines of codes) for build backward propagation algorithm use **in-memory compilation**
- **PyTorch** (by Facebook, 1.7M lines of codes) use **dynamic** building of **computation tree** (more flexible: allow ***if*** and ***while*** constructions)

COSMO-Ru postprocessing at RHM

COSMO-Ru postprocessing (PP) based on PyTorch. Stages:

1. DL systematic correction (at SYNOP stations points) – *operational*

2. Addition DL correction based on the forecasts archive at SYNOP stations points (including lagged forecasting) – *operational*

3. Interpolation correction's increments to grid – *in develop*

Features:

- Cross-platform Windows/Linux and hardware independent CPU/GPU
- 32/64bit training, 16 (GPU only, ~1.3x faster 32bit)/32/64bit inference

Not yet implemented:

- MPI support
- Retuning parameters "online"
- Mixed (16bit arguments & 32bit parameters) precision learning

DL nonlinear Systematic Correction (DLSC)

Let $\vec{\Psi}(t,\tau,\vec{x})$ be the COSMO model surface forecast from t to $t+\tau$ at point $\vec{x}$, containing: 1.temperature T_{2m}; 2.dew point Td_{2m}; 3.PMSL P_0; 4.wind speed $\vec{U}_{10m}$; 5.max wind speed V10MAX G_{10m}; etc.

If we have SYNOP measurements $X_{fact}(t)$ (and increments $BIAS_X(t-\delta t,\tau)=X_{fact}(t-\delta t+\tau)-X(t-\delta t,\tau)$) of weather parameter X at the forecast point $\vec{x}$ we can apply the SC:

$$X_{SC}(t,\tau)=X(t,\tau)+\frac{\sum_{\delta t=\lfloor\tau\rfloor}^{L}\exp(w_{\delta t})\,BIAS_X(t-\delta t,\tau)}{\sum_{\delta t=\lfloor\tau\rfloor}^{L}\exp(w_{\delta t})},$$ where $L = 35$ days

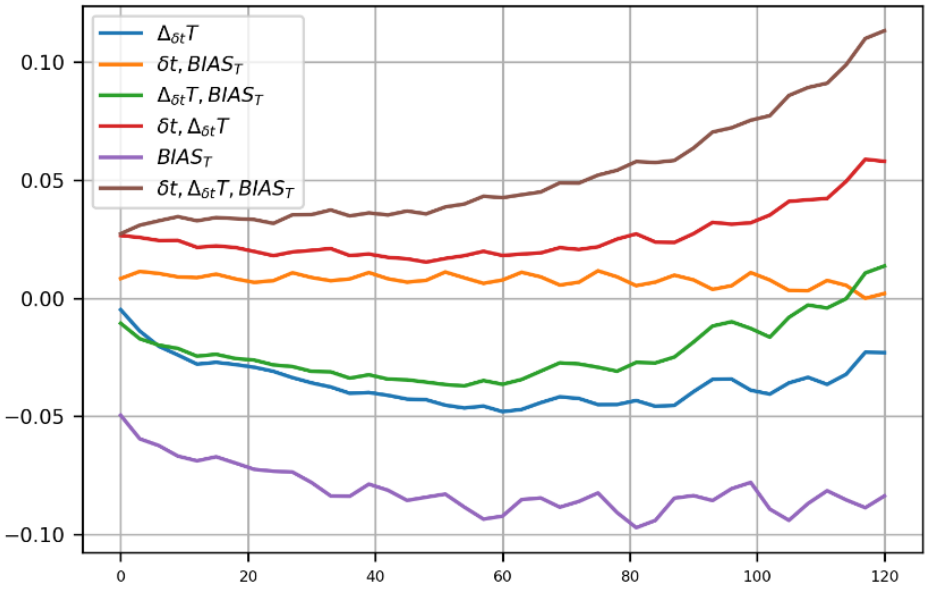
<u>*Classical approach:*</u> the weight tensor $w_{\delta t}$ is $w_{\delta t}\equiv 0$ or $w_{\delta t}=-\lambda\delta t,\lambda=const$.

<u>*DLSC approach:*</u> we can optimize the weights $w_{\delta t}$ directly, for example as:

$$w_{\delta t}=w_{NeuralNet}\left(\delta t,\tau,\vec{\Psi}(t,\tau)-\vec{\Psi}(t-\delta t,\tau),BIAS_X(t-\delta t,\tau)\right).$$

If $w_{\delta t}$ depend on $BIAS_X(t-\delta t,\tau)$ then DLSC is **nonlinear**.

MAE gains of DLSC for T_{2m} correction



Mean absolute error (MAE) gain depends on lead time τ for different sets of arguments of neural net w_{NN}. The 0 corresponds w_{NN} depends only δt: $w_{\delta t} = w(\delta t)$. Where $\Delta_{\delta t}T = T_{2m}(t,\tau) - T_{2m}(t-\delta t,\tau)$.

The nonlinear DLSC significant better then linear (see brown and red lines).

Addition corrections

We will search additive corrections for various points $\vec{x}$ independently and use K shifts by initial time t and (or) lead time τ:

$$\vec{\Psi}_{NN}(t,\tau) = \vec{\Psi}_{SC}(t,\tau) + \vec{F}\left(\vec{\varphi}(t+\tau,\vec{x}),\tau,\vec{\Psi}_{SC}(t-\Delta t_1,\tau+\Delta\tau_1),\ldots,\vec{\Psi}_{SC}(t-\Delta t_K,\tau+\Delta\tau_K)\right),$$

where $\Delta t_k \geq 0$, $\vec{\varphi}(t+\tau,\vec{x})$ describes time $t+\tau$ and location $\vec{x}$.

If $\Delta t_k = \Delta\tau_k \Rightarrow$ we use an older forecast from $t-\Delta t_k$ to $t+\tau$.

Embedding

The *embedding* is the SYNOP stations table mapping to [0;1], such as:

	E_1	E_2	E_3
10637 Frankfurt	0.34	0.77	0.32
11034 Wien	0.56	0.43	0.18
27612 Moscow	0.14	0.88	0.72
…	…	…	…

The numbers in the table are **additional optimized parameters**

Predictions for addition corrections

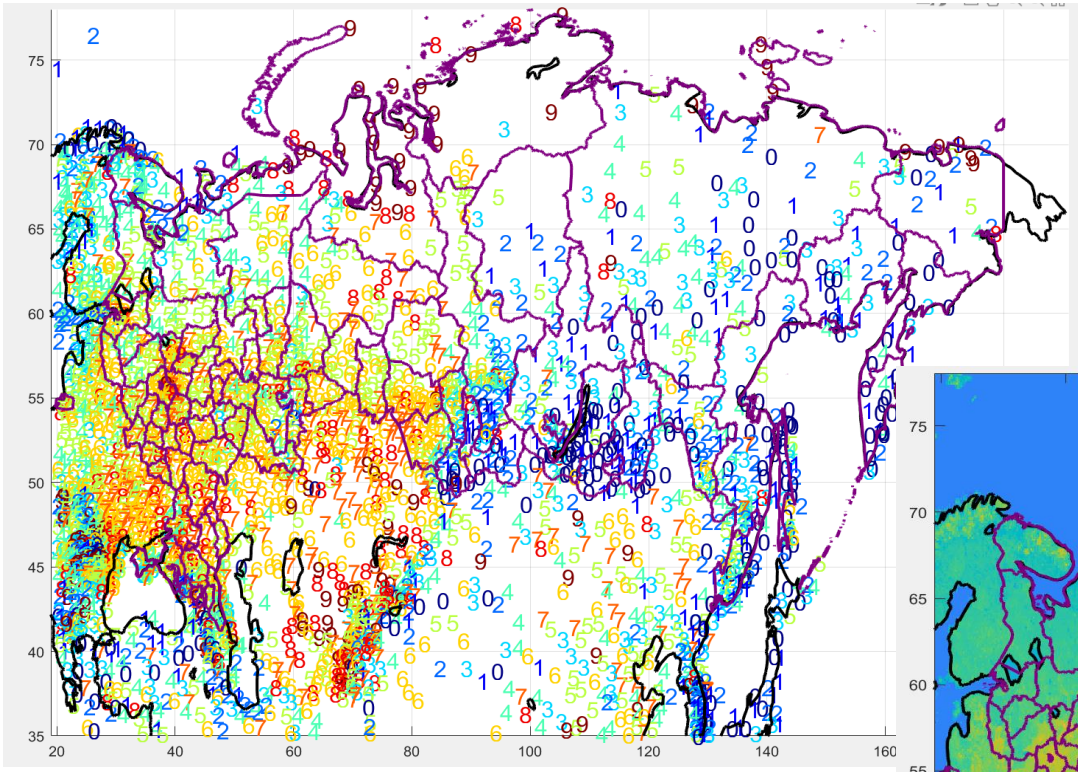
We use N_{old} older forecasts and the weather data for the last $N_{history}$ days:

Options	Predic-tions	List of predictions
$N_{old}=0$ $N_{history}=0$	43	$E_{1,2,3}, \tau, \alpha(t+\tau-3h), \alpha(t+\tau), \alpha(t+\tau+3h), \Delta x, B(t,\tau+\Delta\tau)-B(t,\tau),$ $B(t,\tau), \Delta\tau=-6h,-3h,3h,6h, B=T_{2m}, Td_{2m}, P_0, \vec{U}_{10m}, \lvert\vec{U}_{10m}\rvert, G_{10m}$
$N_{old}=0$ $N_{history}=2$	61	For $N_{old}=N_{history}=0$ and $A(t-\Delta t,\tau+\Delta\tau)-A(t,\tau), \Delta\tau=-6h,0h,6h, \Delta t=24h,48h, A=T_{2m}, Td_{2m}, P_0$
$N_{old}=2$ $N_{history}=0$	87	For $N_{old}=N_{history}=0$ and $B(t-12h,\tau+\Delta\tau)-B(t,\tau), \Delta\tau=6h,12h,18h$ $B(t-24h,\tau+\Delta\tau)-B(t,\tau), \Delta\tau=18h,24h,30h, B=T_{2m}, Td_{2m}, P_0, \vec{U}_{10m}, \lvert\vec{U}_{10m}\rvert, G_{10m}$

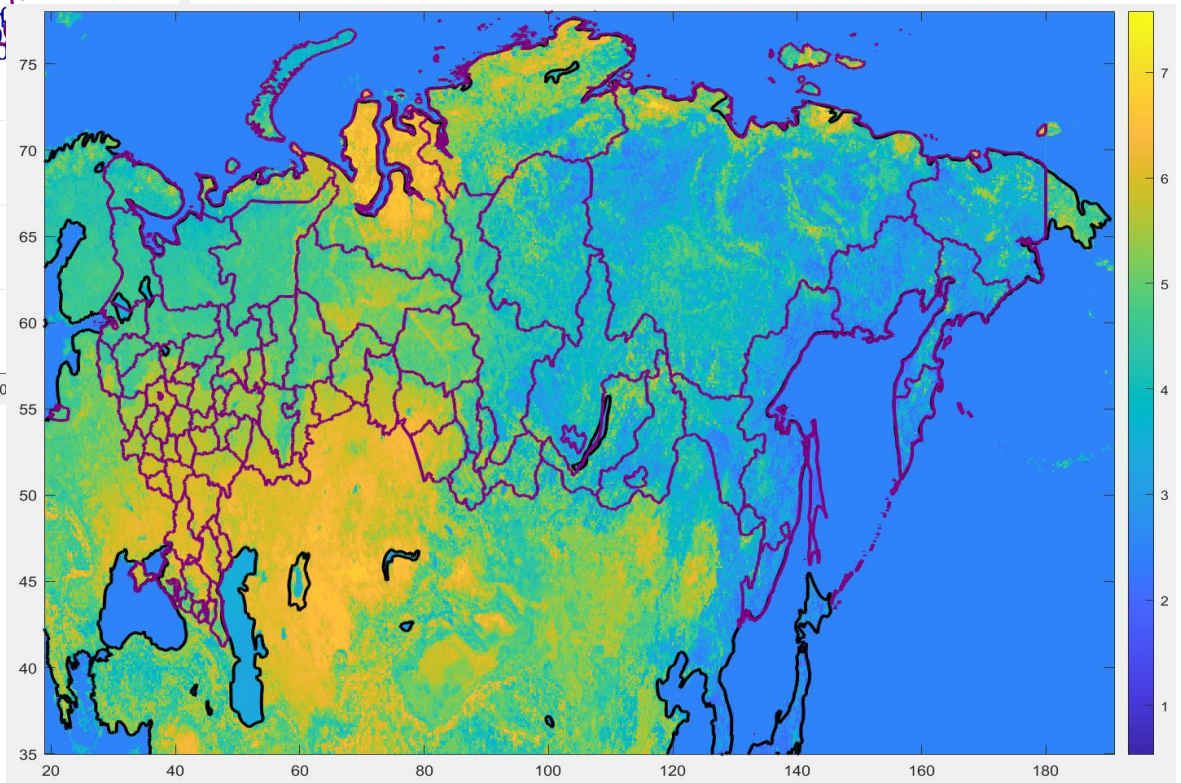
$\alpha(t)$ – the Sun elevation angle (signed); Δx – COSMO grid spacing

The priority row of ($N_{old}, N_{history}$) pairs:
(3,1), (2,1), (1,2), (0,2), (0,0).

The optimized Embedding on maps



ML* embedding's interpretation based on geographic: landcover type, soil type, distance to sea, orographic data



*Use the gradient boosting decision tree (lightGBM library)

Datasets	Training			Testing		
Time range	2017-18	June2019-June2020		August 2020		
COSMO version	5.03			5.06		5.00 urban
Model config.	Ru13-ENA	Ru6-ENA	Ru2-ETR	Ru6-ENA	Ru2-ETR	Ru1-Msk
Grid spacing	13.2km	6.6km	2.2km	6.6km	2.2km	1.0km
Grid size	1000×500	2000×1000	1400×1200	2000×1000	1400×1200	180×180
Initial time	00, 12 GMT			00, 06, 12, 18 GMT		
Forecast length	99h	120h	48h	120h	48h	48h
SYNOP stations	3771	2802	778	2802	778	39
Forecasts*	1.3×10^{8}	6.3×10^{7}	9.2×10^{6}	9.1×10^{6}	1.4×10^{5}	6.1×10^{4}

* The number of forecast–SYNOP observation station pairs in the dataset

Neural networks (NN) computational costs

Our training dataset contain ~2.1×10^8 realizations.
The FLOPs number $C_{operational}$ for the operational processing for one initial time and $C_{training}$ for the NN training are related as

$$C_{training} \approx 2.2 N_{init} N_{epoch} C_{operational}$$

where N_{init} is the number of initial times in the training dataset, N_{epoch} is the number of epochs for the optimization of parameters $\vec{\theta}_{\vec{\mu}}$.

For a typical case, $N_{init} = 365\times4$, $N_{epoch} = 20$ the time ratio is ~64 000:

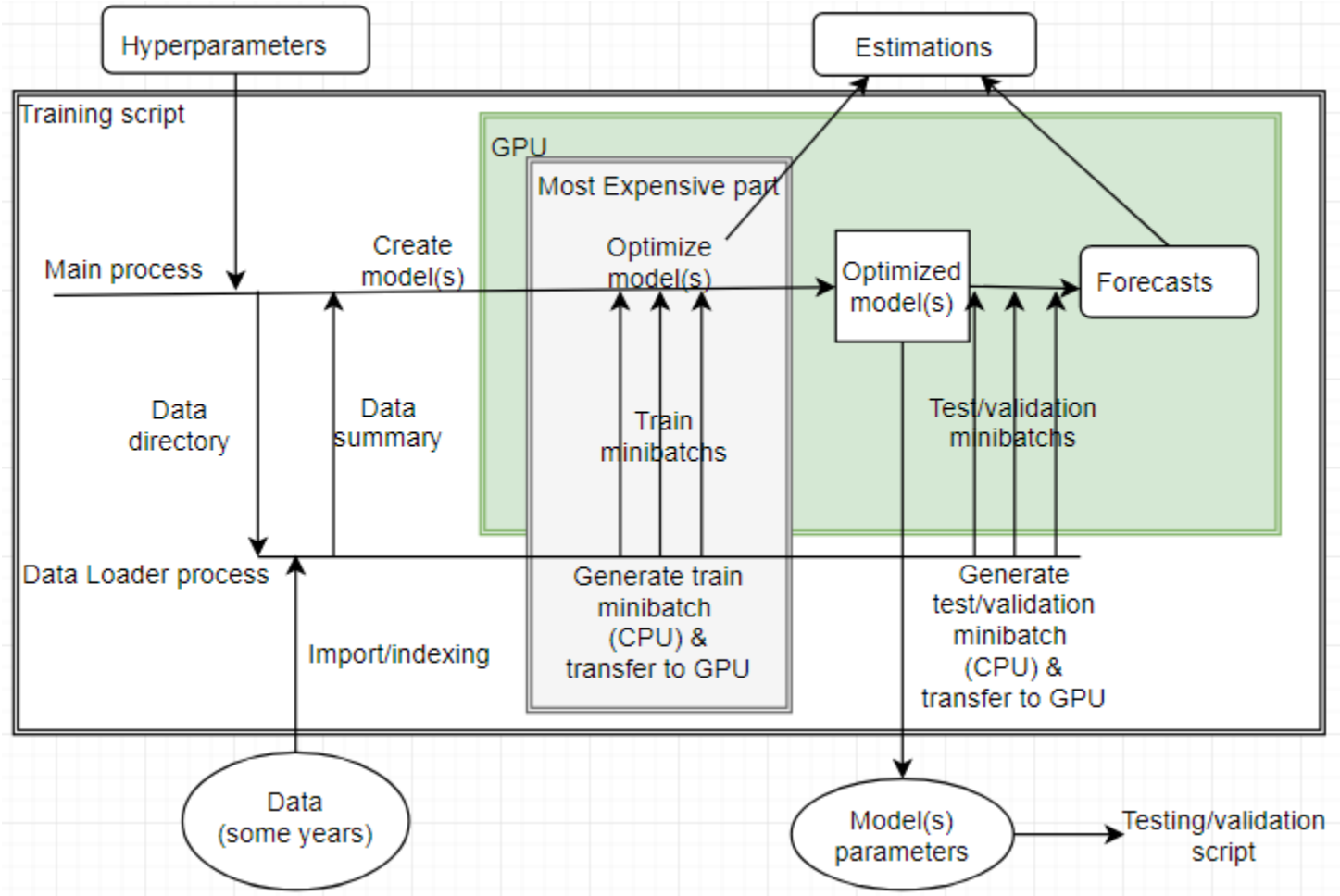
<u>100 seconds</u> for operational processing $\Rightarrow$ <u>75 days</u> for training

The training with hyperparameters tuning is more expensive!

Big data training pipeline

For big data processing we must split the dataset to minibatchs.

The computations are most efficient if CPU loads and prepares the minibatch and GPU processes it



Training pipeline optimization

For PC with a 8 core CPU (~270 GFLOP/s) and
Nvidia RTX 2070 GPU (~6500 GFLOP/s)

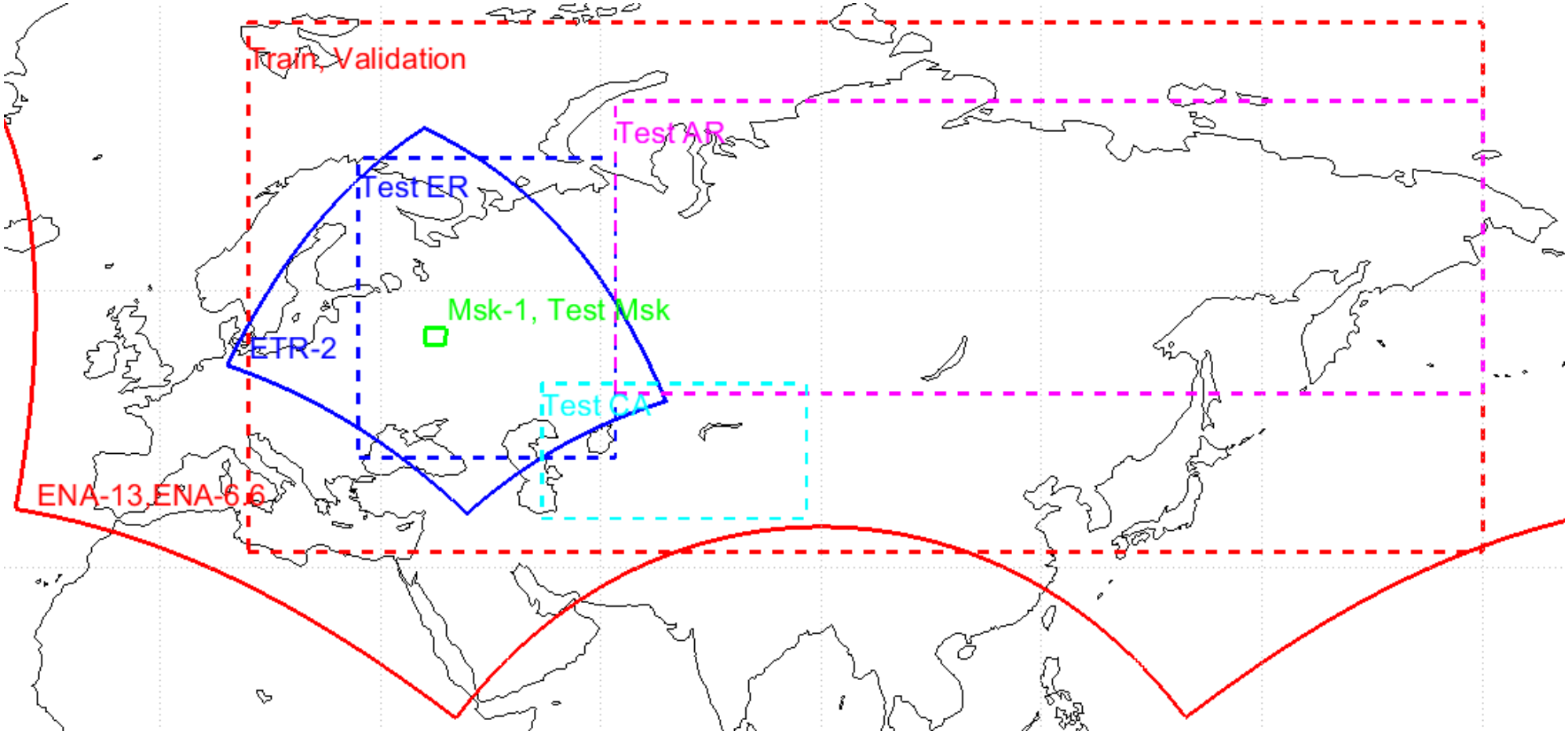
	CPU utilization	GPU utilization	Time per training epoch per model*
Baseline CPU	100%	0%	13 min 41 s
Baseline GPU	1 core	4%	15 min 12 s
+Parallel Data Loader	100%	30%	2 min 4 s
+Training 10 models simultaneously**	35%	70%	55 s

*Model with ~10 000 parameters, dataset volume ~6×10^7

**We need to optimize several models for tuning hyperparameters

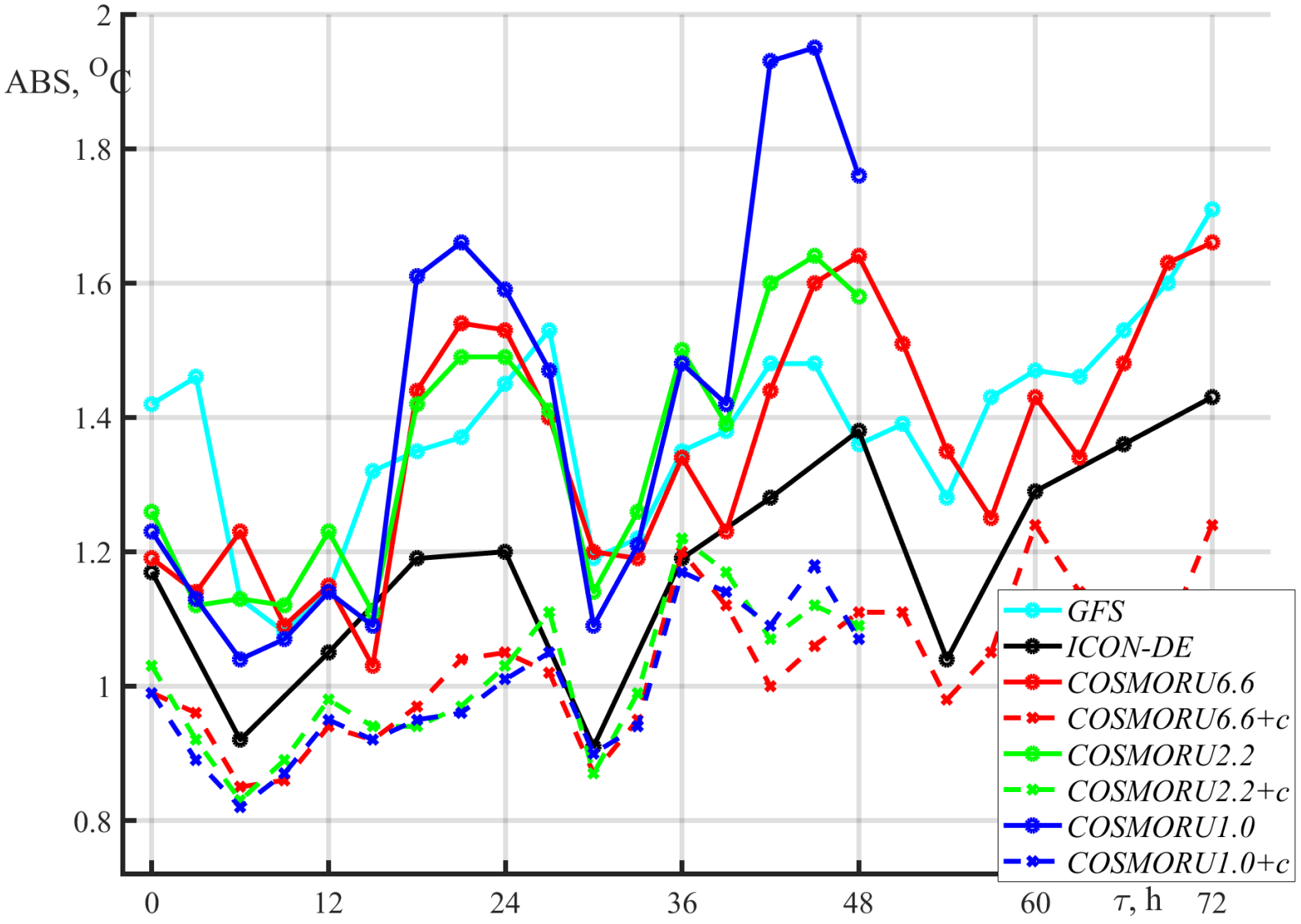
GPU utilization is up to 97% for model with >10^6 parameters

COSMO-Ru models, training and testing regions

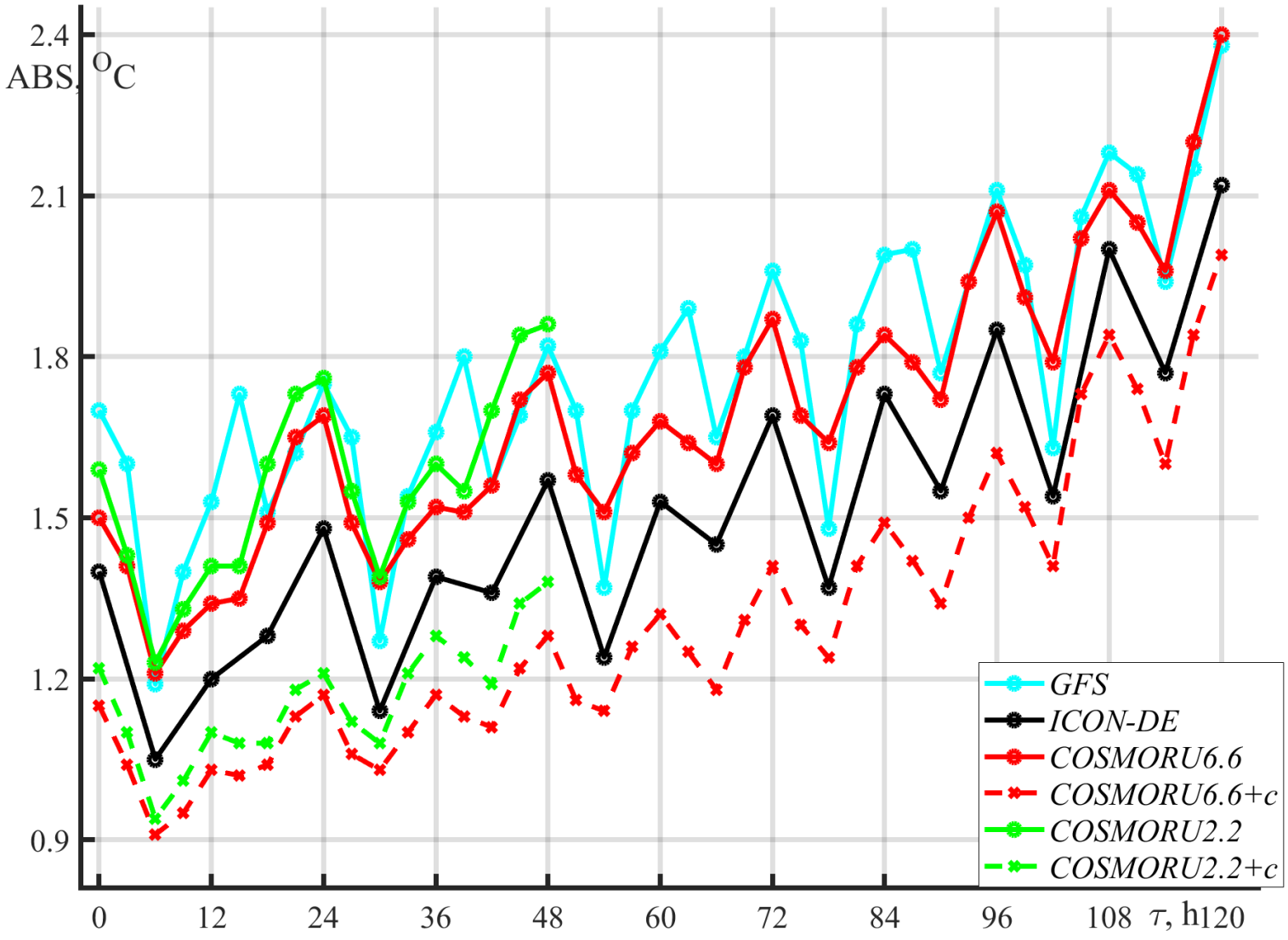


The test results for postprocessing (PP) forecasts with **initial time 00 GMT** for **August 2020** are presented below

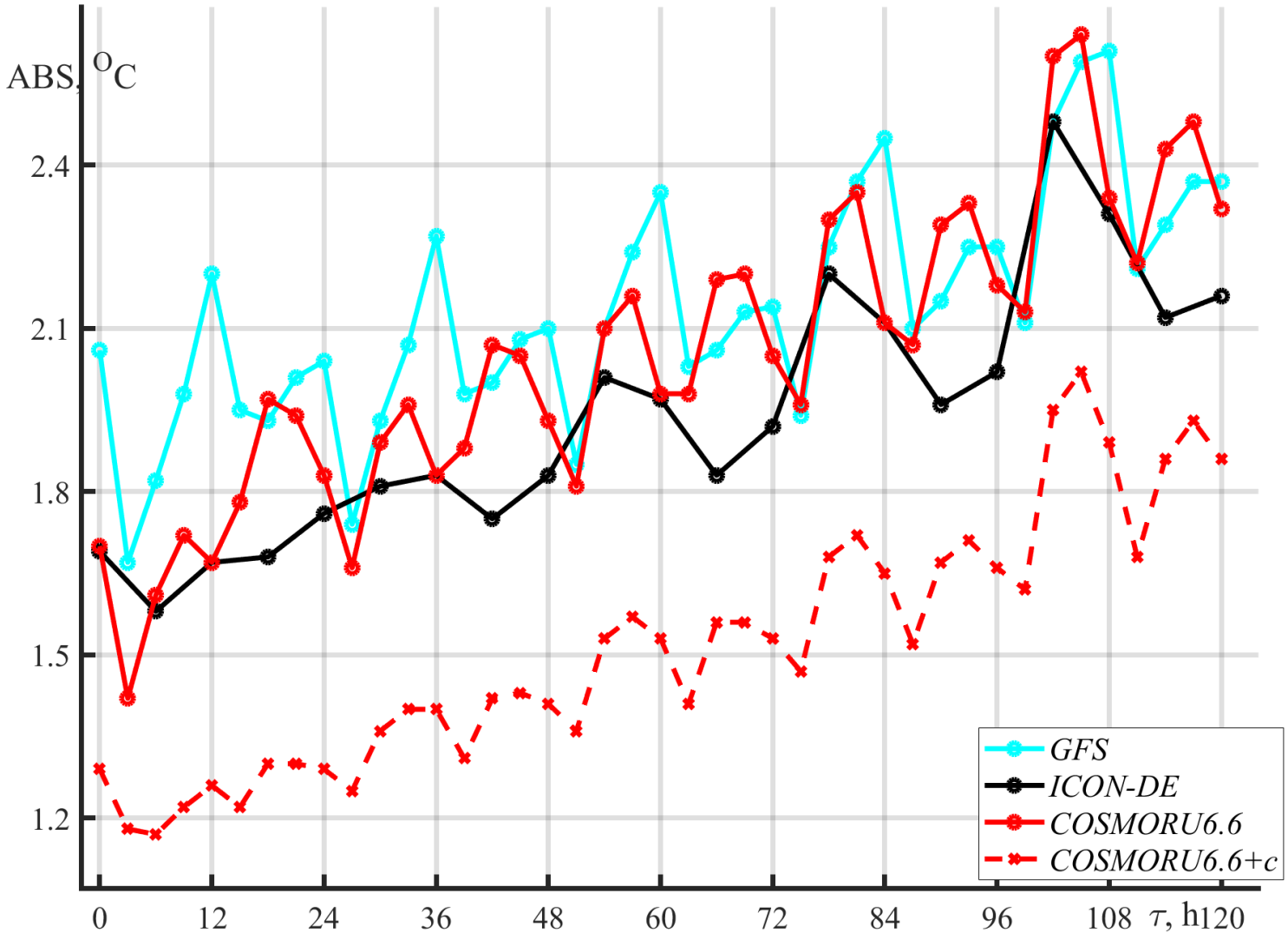
Testing T_{2m} PP in Moscow region



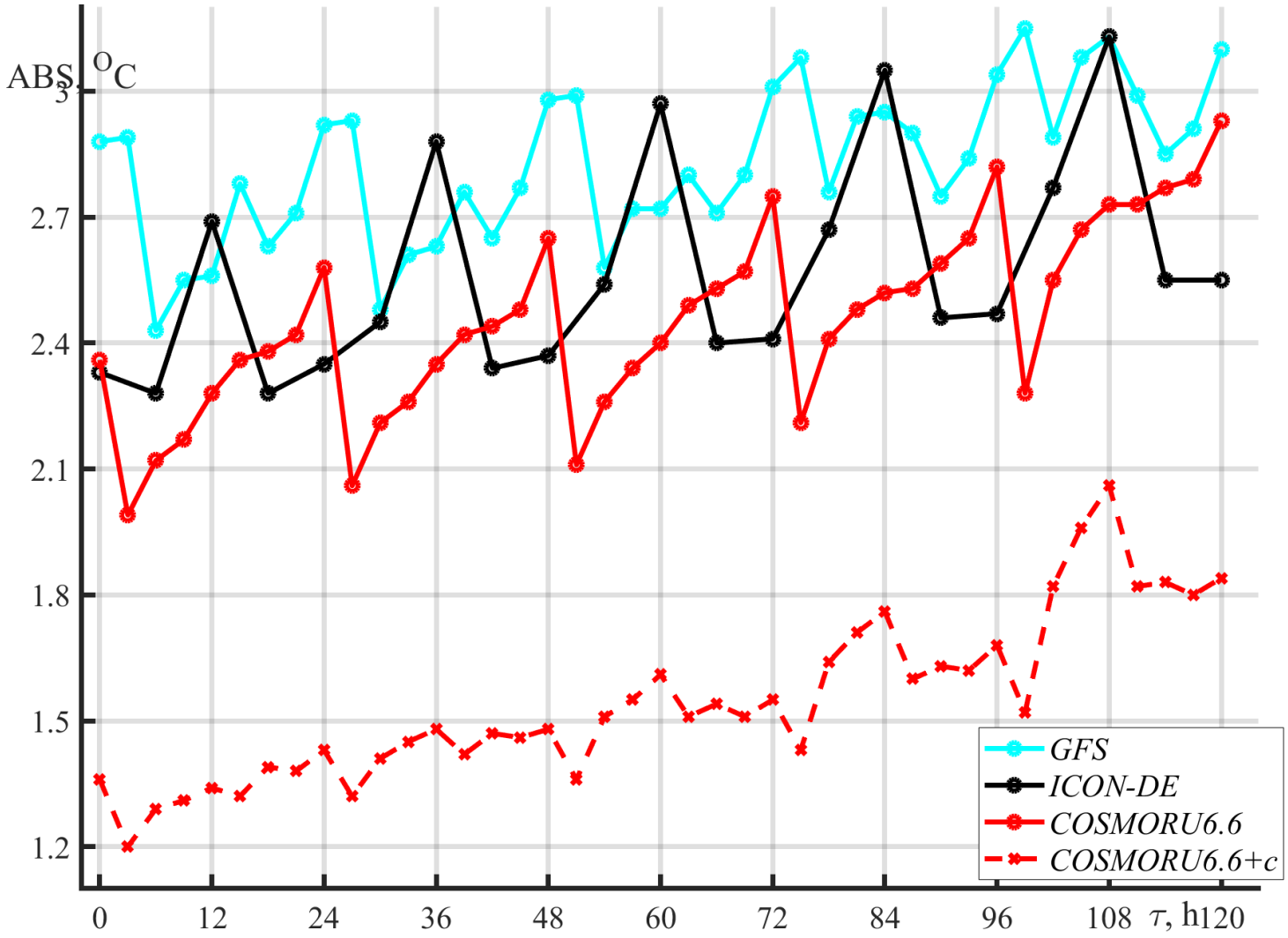
Testing T_{2m} PP in European Russia



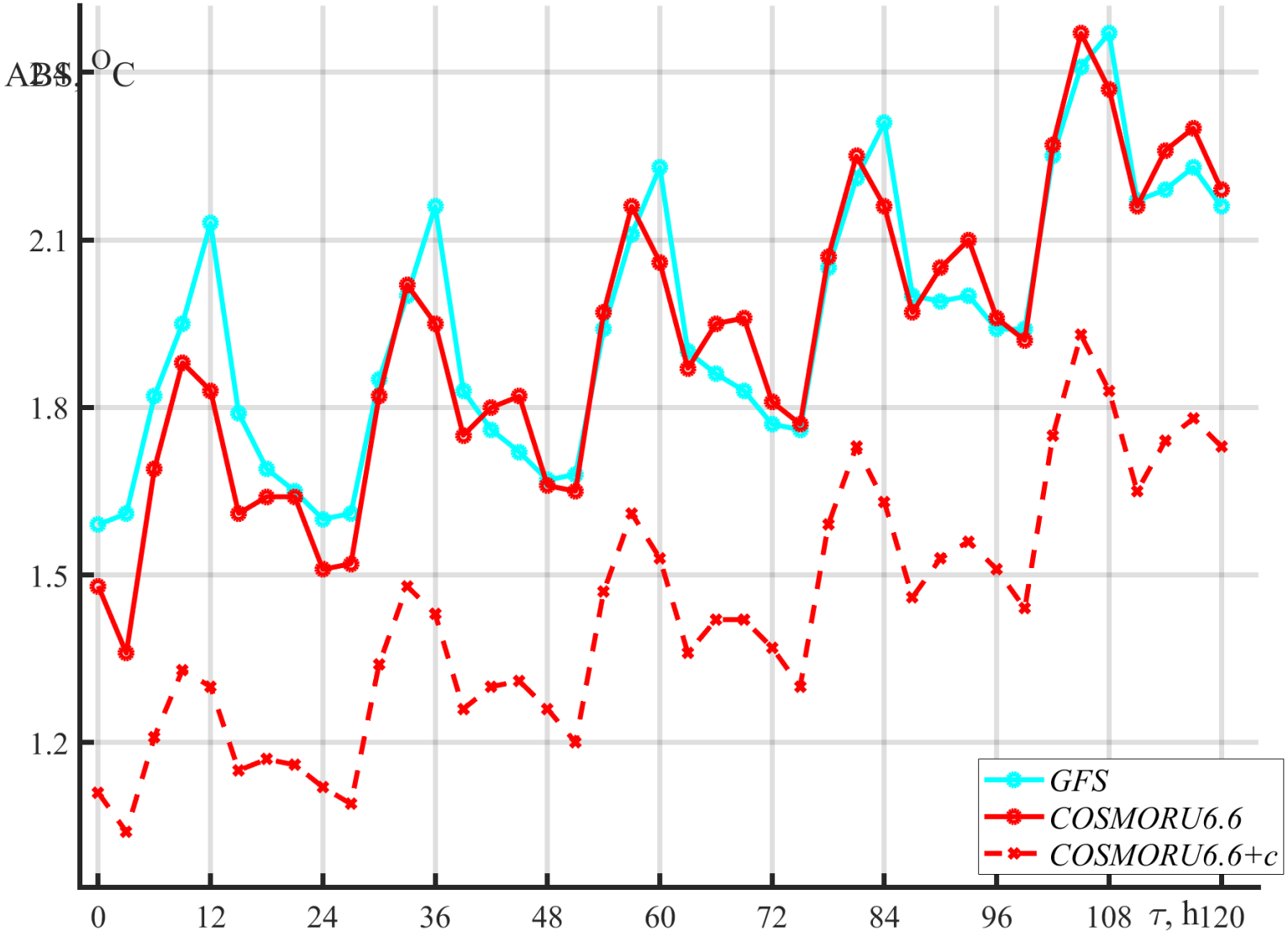
Testing T_{2m} PP in Asian Russia



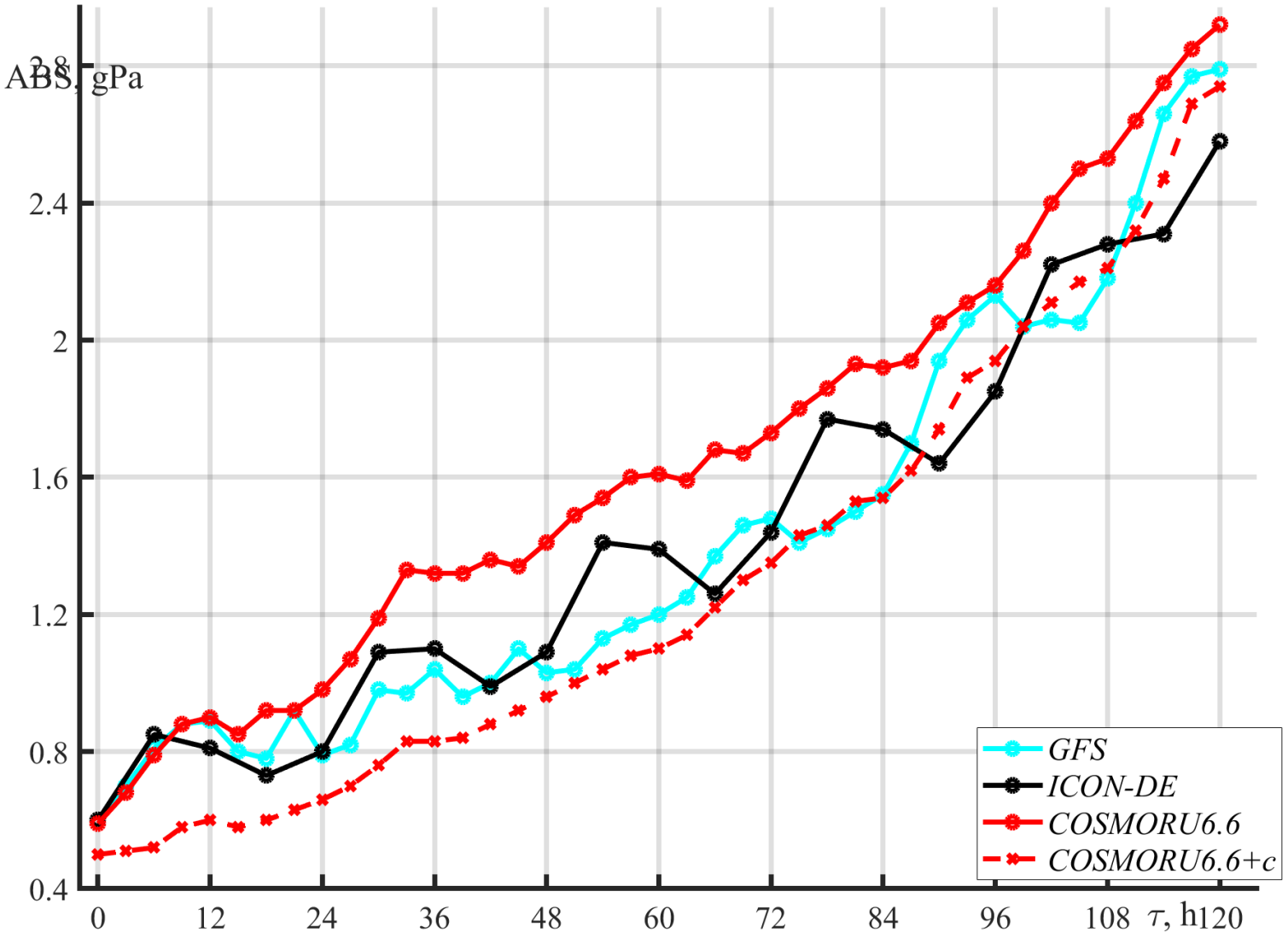
Testing T_{2m} PP in Central Asia



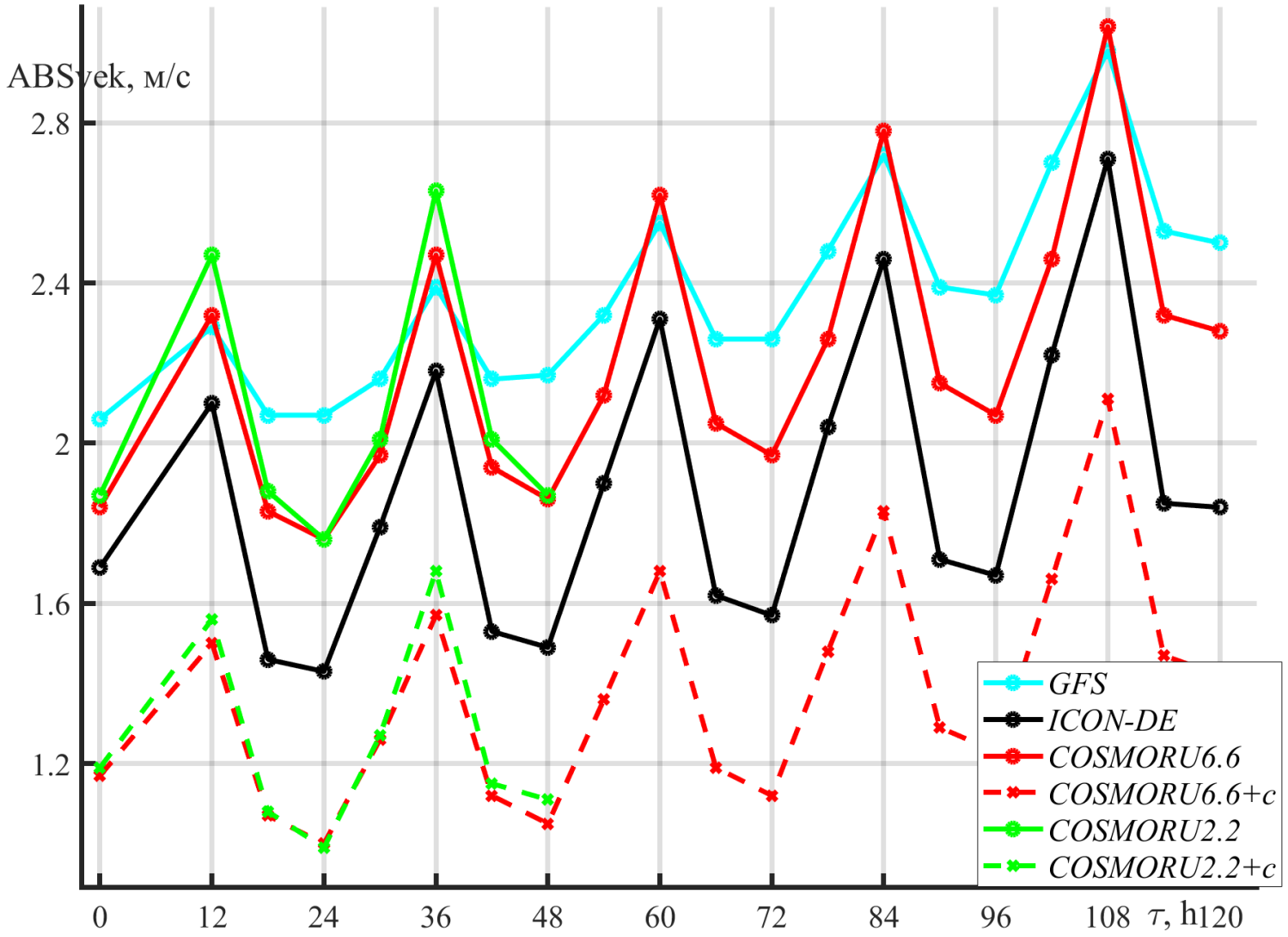
Testing dew point Td_{2m} PP in Asian Russia



Testing PMSL P_0 PP in Asian Russia

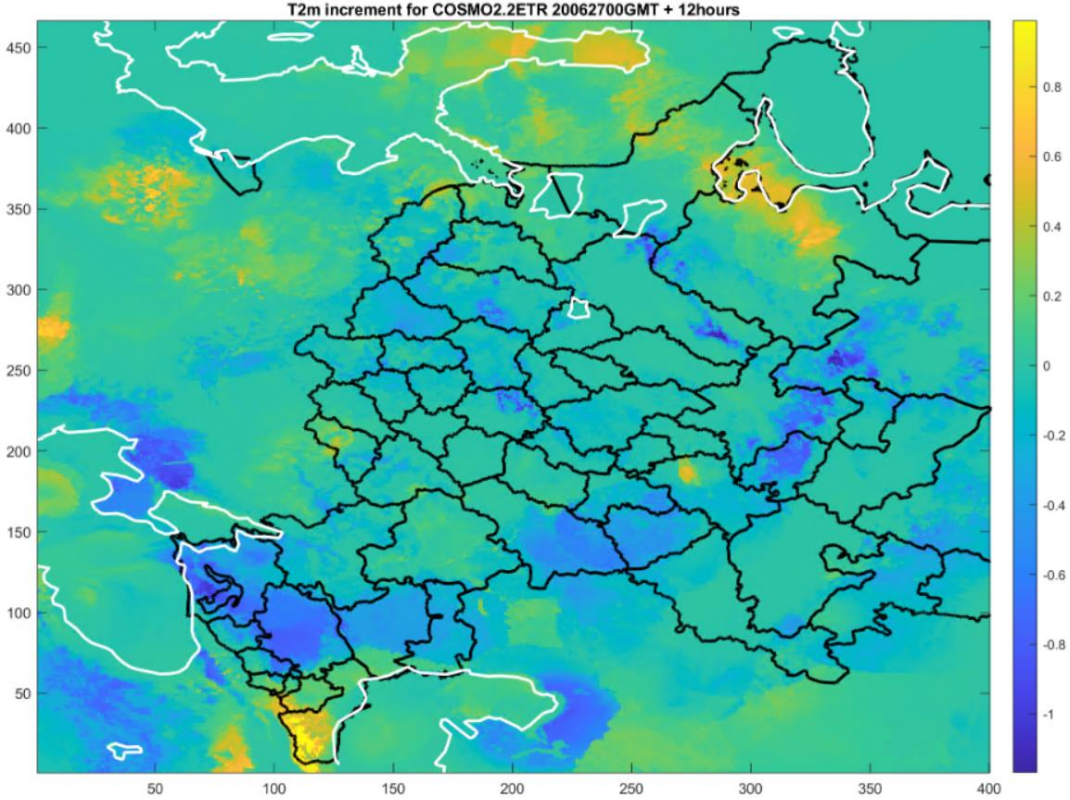


Testing 10m wind speed PP in European Russia

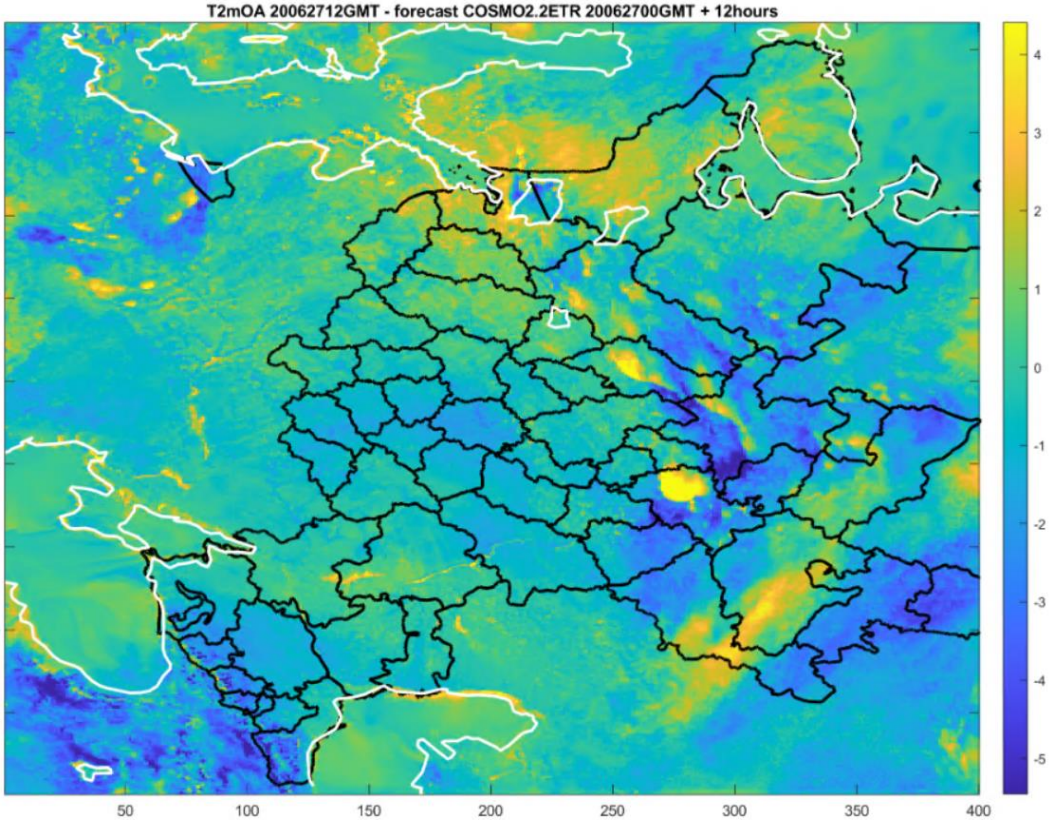


Example T_{2m} PP increment field for COSMO-Ru2.2ETR

Post-processing increment field (0GMT+12h)



Feature initial (12GMT+0h) – forecast (0GMT+12h)



Correlation = 27.7%

MAE decreased by 0.002^{O}C after two interpolations (SYNOP -> grid -> SYNOP)

Conclusions

1. The application of DL methods can improve surface weather forecasts. The DL processed forecasts have the accuracy close to the accuracy of raw forecasts with **2-2.5 days shorter lead time**
2. The **nonlinear** systematic correction **better** then linear
3. **Same method** can improve forecasts issued by various COSMO model configurations **with 13.2km, 6.6km, 2.2km, 1km grid spacings**
4. The embeddings approach is helpful for take into account the local parameters

Plans

1. The precipitation's probability distribution forecasting
2. Apply to ICON model
3. Use aerological data

Thank you for attention!

About program realization

Python 3.7.5, PyTorch 1.3.1
initialization: Xavier uniform
loss function: Huber loss («SmoothL1» in PyTorch)
epochs: 20
optimizer: «AdamW», initial learning rate: 0.01
weight decay: 0.005
scheduler: «LambdaLR», lambda: $(1+2\cdot epoch)^{-1}$