

Meteorological Super-Resolution vs Wind Representations

Gruppo 21
Marzia De Maina, Matteo Galiazzo, Federica Santisi

Alma Mater Studiorum - Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria (DISI)

- Evaluate the impact of **different wind representations** on neural-based super-resolution for meteorological fields.
- Compare two **wind encodings**:
 - Orthogonal components u_{10}, v_{10}
 - Polar form: speed and direction (degrees from north)
- Use **ERA5 (30 km)** and **VHR-REA (2.2 km)** datasets for training and testing.
- Assess which representation yields better performance and qualitative reconstruction.

Cartesian Components

- u_{10}, v_{10} : zonal and meridional wind at 10m height
- Directly match ERA5 outputs

Polar Encoding

- speed = $\sqrt{u_{10}^2 + v_{10}^2}$
- direction = $(180 + \frac{180}{\pi} \arctan 2(-u_{10}, -v_{10})) \text{ mod } 360$

Motivation: Representation may affect neural network learning and reconstruction quality.

Super-Resolution

Upscaling low-resolution fields to high-resolution targets using deep learning. This technique involves reconstructing high-resolution images from coarser data, with the aim of capturing finer and more complex atmospheric structures.

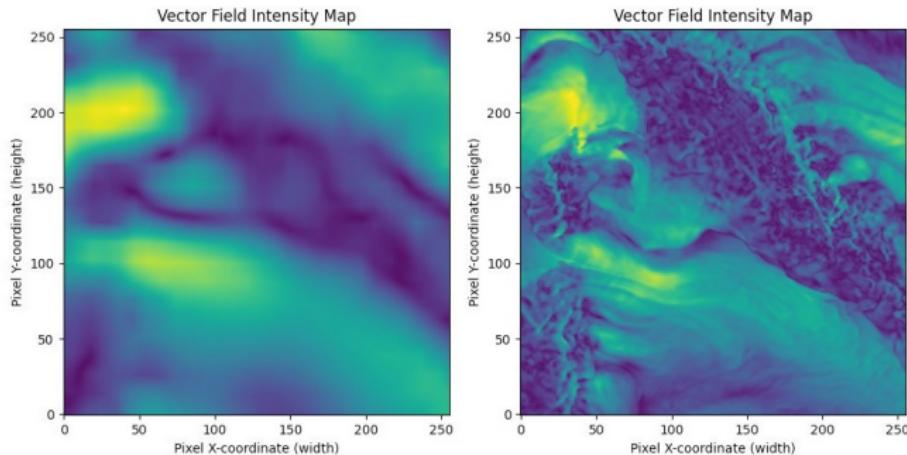


Figure: Super-resolution in the best case

U-Net Architecture

Encoder-decoder with skip connections for image-to-image tasks.

- Residual blocks to ease training.
- Multiscale features preserved via concatenation.

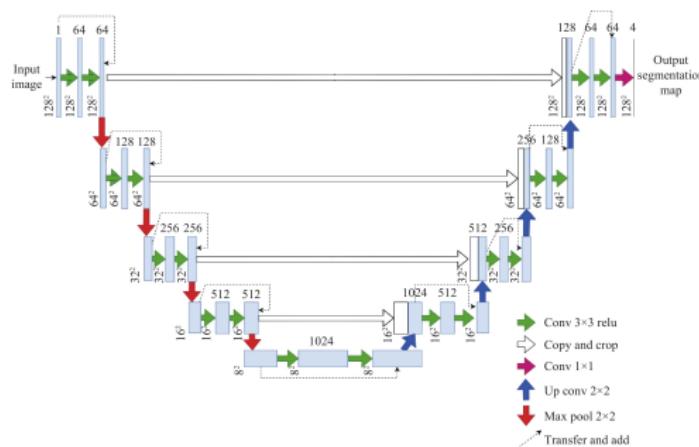


Figure: U-Net Standard Architecture

For the purpose of this project, the wind field representations are derived from the ERA5 and VHR–REA datasets.

- **ERA5** is a global reanalysis dataset developed by ECMWF, providing hourly data since 1950 at a 0.25° (~ 31 km) resolution.
- **VHR–REA** is a downscaled regional reanalysis for Italy based on the COSMO model, with a resolution of 2.2 km.

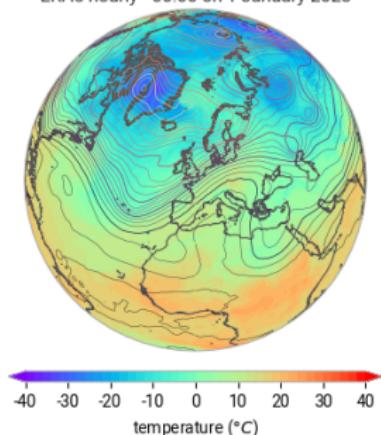
Both datasets are aligned temporally (06, 12, 18, 00 UTC) and spatially through reprojection.

ERA5 acts as the low-resolution input; VHR–REA is used as the high-resolution target for training.

Datasets and Preprocessing

- ERA5 is the fifth-generation global reanalysis dataset produced by ECMWF under the Copernicus Climate Change Service (C3S). It replaces the previous ERA-Interim product, offering higher accuracy, finer temporal and spatial resolution (0.25°), and consistent uncertainty estimates.
- Based on a 4D-Var data assimilation scheme, ERA5 integrates a vast range of observational data, including satellite, radiosonde, buoy, and aircraft measurements. It covers the atmosphere, land surface, and ocean waves with hourly updates and is widely used for climate studies, energy forecasting, and risk assessment.
- In this project, we focus on the surface wind fields, specifically the u_{10} and v_{10} components, which serve as low-resolution inputs for the super-resolution model.

850 hPa temperature and 500 hPa geopotential
ERA5 hourly - 00:00 on 1 January 2023



Datasets and Preprocessing

VHR–REA (Very High Resolution ReAnalysis) is a dataset produced by CMCC by dynamically downscaling ERA5 with the COSMO regional model. The dataset has 2.2 km resolution, explicitly resolving convection processes important in complex terrain.

This high-resolution data supports regional applications such as hydrological forecasting, renewable energy planning, and urban climatology.

- Target resolution for training a super-resolution model.
- Data span: one full year, 4 times daily over Northern Italy.

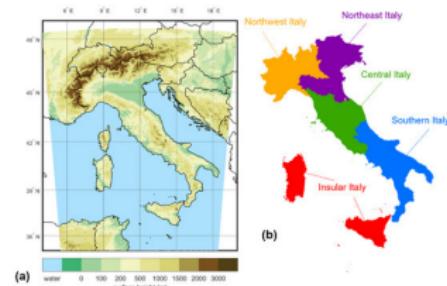


Figure: VHR–REA spatial coverage over Northern Italy

Dataset and Preprocessing

Spatial Alignment & Patch Extraction

- Center crop to 224×224 pixel patches
- Temporal synchronization verification
- Channel-wise data stacking ($u10, v10$ components)
- Custom PyTorch Dataset implementation

Data Pipeline Architecture

- ItalyWeatherDataset class for paired data loading
- Automatic temporal alignment checking
- Flexible normalizer integration
- Memory-efficient batch processing

Quality Assurance

- Index bounds verification
- Consistent tensor dimensions
- Error handling for misaligned datasets

```
def extract_region(self, data_tensor):
    patch_h, patch_w = (224, 224)
    _, h, w = data_tensor.shape
    x = (w - patch_w) // 2
    y = (h - patch_h) // 2
    return data_tensor[:, y:y+patch_h,
                      x:x+patch_w]

def __getitem__(self, idx):
    era_slice = self.era5_dataset.
               isel(valid_time=idx)
    era_arrays = [era_slice[v].
                  values
                  for v in self.
                  ERA5_VARIABLES]
    era_tensor = torch.from_numpy(
        np.stack(era_arrays, axis=0)
        .float())
    era_tensor = self.extract_region
    (era_tensor)

    if self.era5_normalizer:
        era_tensor = self.
                    era5_normalizer.
                    normalize(era_tensor)
```

Normalization & Coordinate Transformation

MinMax Normalization Strategy

- Training-based statistics computation
- Per-channel normalization across spatial dimensions
- Feature range scaling to [0, 1]
- Epsilon handling for division-by-zero cases
- Serializable normalizer objects

Coordinate System Transformation

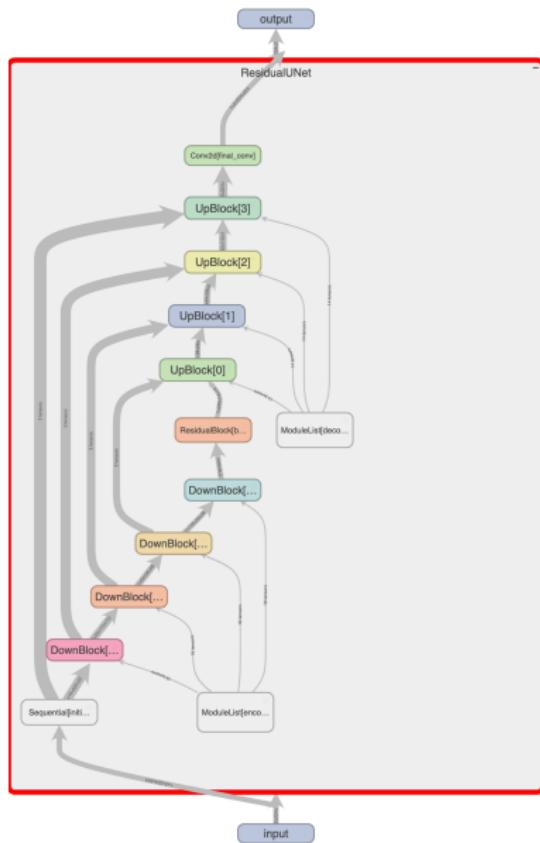
- Option 1: Cartesian coordinates (u , v)
- Option 2: Polar coordinates (magnitude, direction)
 - $magnitude = \sqrt{u^2 + v^2}$
 - $direction = 180 + \frac{180}{\pi} \arctan 2(-u, -v) \bmod 360$

Dataset Management

- 80-20 train-test split with fixed seed (42)
- Separate normalization preservation
- Batch processing with memory optimization

```
class MinMaxNormalizer:  
    def compute_stats(self, data_tensor):  
        # Min/max per channel across (N,H,W)  
        dims  
        self.min_val = data_tensor.amin(  
            dim=(0, 2, 3), keepdim=True)  
        self.max_val = data_tensor.amax(  
            dim=(0, 2, 3), keepdim=True)  
  
        # Handle division by zero  
        diff = self.max_val - self.min_val  
        epsilon = 1e-7  
        self.max_val[diff < epsilon] = \  
            self.min_val[diff < epsilon] +  
            epsilon  
    def normalize(self, x):  
        return (x - self.min_val) / \  
            (self.max_val - self.min_val)  
  
# Coordinate transformation to polar  
if COORDINATES == "1":  
    u_squared = tensor[0, :, :]**2  
    v_squared = tensor[1, :, :]**2  
    magnitude = torch.sqrt(u_squared + v_squared)  
    )  
    direction = (180 + (180/math.pi) *  
                torch.atan2(-u_squared, -  
                            v_squared)) % 360  
    tensor = torch.stack([magnitude, direction],  
                        axis=0)
```

Model and hyperparameters - General differences



Interesting points in our implementation:

- Conv2d with stride=2 for downsampling.
- BatchNorm to reduce internal covariate shift. This improves speed and stability of the training.

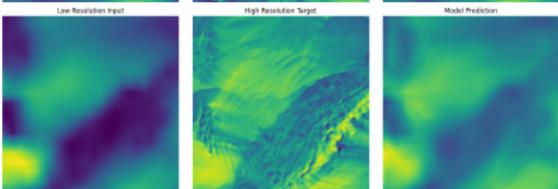
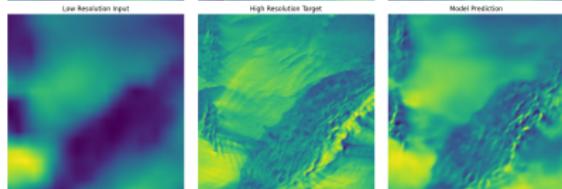
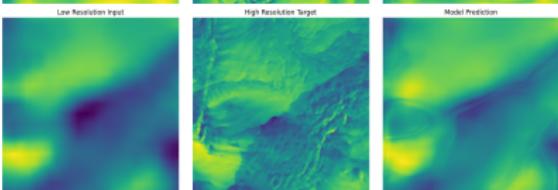
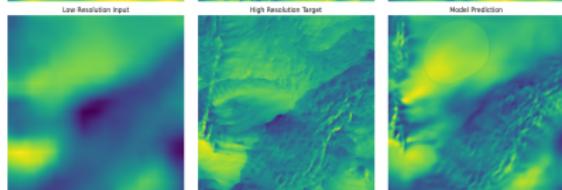
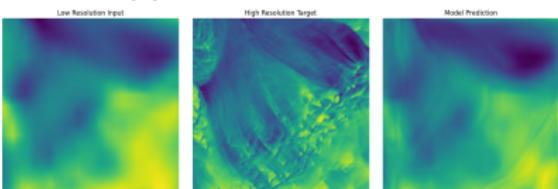
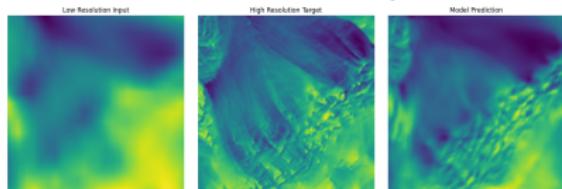
But there's more...

Network Type	Parameters Count	Training Time	Test Loss	Test SSIM
Transposed Conv	13,041,922 (13.0 M)	54:03 16.22s/it	0.19775	0.70910
Bilinear Interp	3,607,586 (3.6 M)	08:04 2.42s/it	0.17166	0.74983

Table: Comparison of bilinear interpolation vs transposed convolution.

Model and hyperparameters - Loss

Even the best model couldn't get decent results with the suggested MSE loss, so we adopted a combined loss approach



L1 + SSIM loss. SSIM: 0.7556

MSE loss. SSIM: 0.7022

Model and hyperparameters - Loss

```
class L1SSIMLoss(nn.Module):
    def __init__(self, alpha=0.85, ssim_window_size=11, ssim_data_range=1.0, ssim_channel=1):
        super(L1SSIMLoss, self).__init__()
        self.alpha = alpha
        self.l1_loss = nn.L1Loss() # Mean Absolute Error
        self.ssim_loss_fn = SSIMLoss(window_size=ssim_window_size, data_range=ssim_data_range, channel=ssim_channel)

    def forward(self, y_pred, y_true):
        ssim_val_loss = self.ssim_loss_fn(y_pred, y_true)
        l1_val_loss = self.l1_loss(y_pred, y_true)

        combined_loss = self.alpha * ssim_val_loss +
                        (1 - self.alpha) * l1_val_loss
        return combined_loss
```

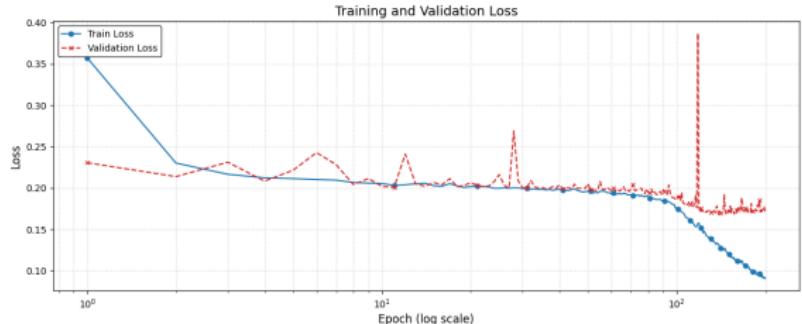
Model and hyperparameters - Hyperparameters

Channels	Epochs	Loss	Batch Size	LR	LR Scheduler	Weight Decay	Test Loss	Test SSIM
[32, 64, 128]	200	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.19090	0.71990
[16, 32, 64, 128]	200	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.17700	0.74170
[16, 32, 64, 128, 256]	300	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.16768	<u>0.75555</u>
[16, 32, 64, 128, 256]	300	MSE + SSIM	8	1.00E-03	NO	1.00E-05	<u>0.15464</u>	<u>0.75391</u>
[16, 32, 64, 128, 256]	300	MSE	8	1.00E-03	NO	1.00E-05	0.00327	0.70218
[16, 32, 64, 128, 256, 512]	300	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.19207	0.71774
[16, 32, 64, 128, 256, 512]	300	L1 + SSIM	8	1.00E-03	YES	1.00E-05	0.19207	0.71774

Table: Performance comparison of different network configurations. Best and runner-up models are highlighted in bold and underline, respectively.

Local Training - Loss and SSIM curves

Training curves for the best model on local data (loss and SSIM):

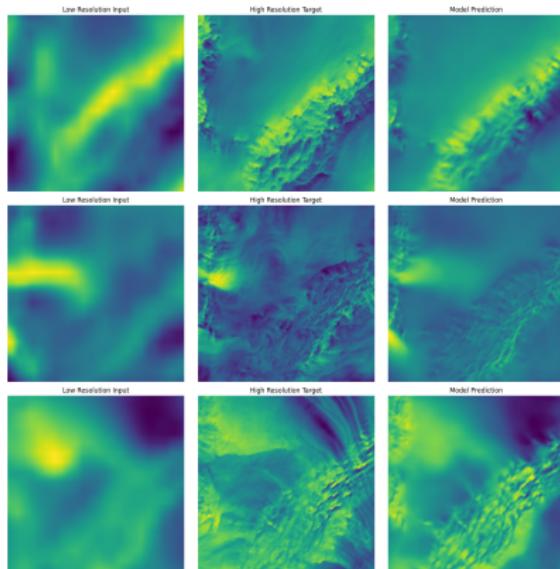


Loss curve

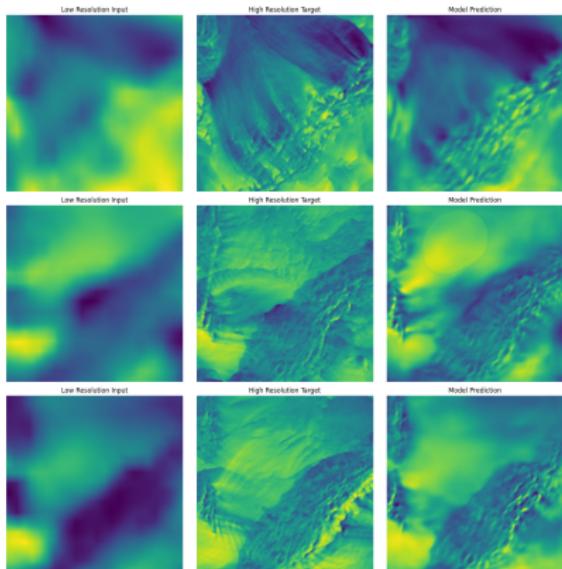


SSIM curve

Colab Training - Difference with Local data



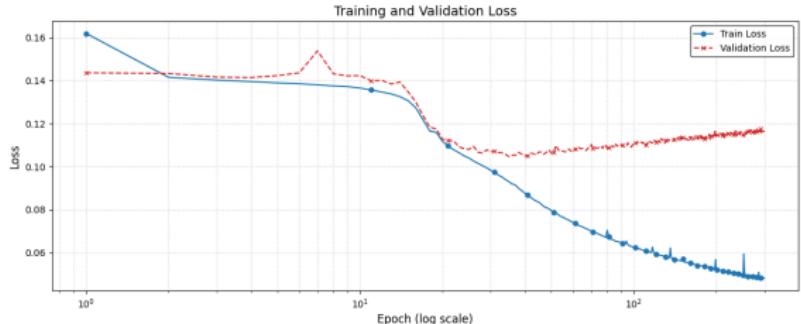
Colab data. SSIM: 0.84483



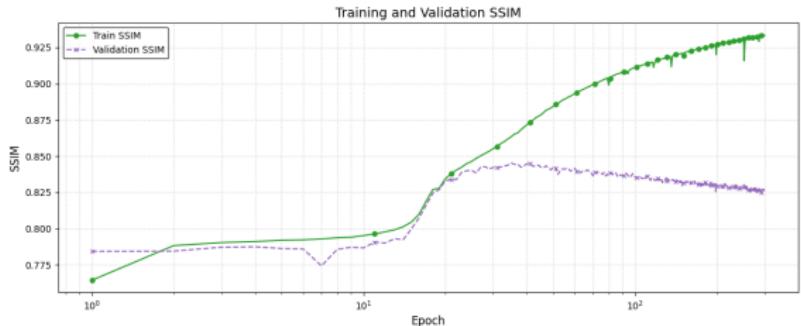
Local data. SSIM: 0.7556

Colab Training - Loss and SSIM curves

Training curves for the best model on colab data (loss and SSIM):

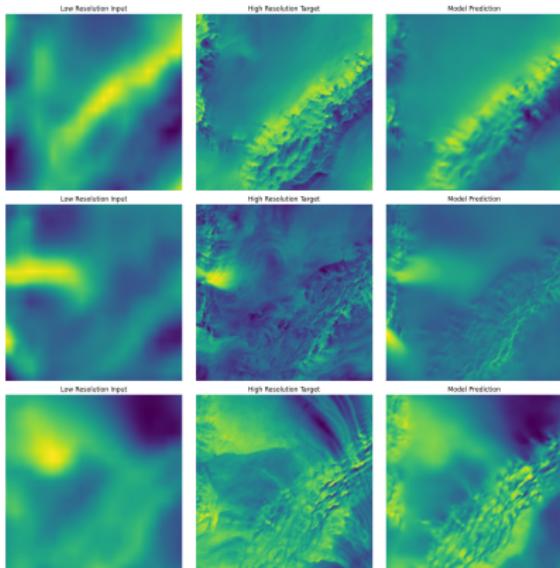


Loss curve

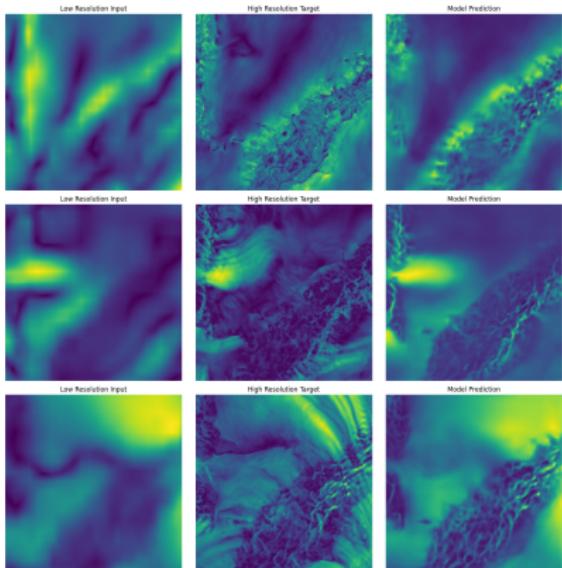


SSIM curve

Colab Training - Vectorial data vs Directional data



Vector data. SSIM: 0.84483



Directional data. SSIM: 0.48154

Comparison between Vectorial and Directional data

Dataset	Test Loss	Test SSIM
Local Vectors	<u>0.16610</u>	<u>0.75830</u>
Local Direction	0.49120	0.38960
Colab Vectors	0.10470	0.84535
Colab Direction	0.42873	0.48154

Table: Comparison of test loss and SSIM between vectorial and directional data

Directional data underperformed vs. vector-based representations

Societal Impact

More accurate high-resolution forecasts support *agricultural management, climate disaster preparedness and the coastal/tourist community.*

Next steps

- Check whether the $L1 + SSIM$ loss still favors accurate reconstruction in complex areas with clear structures, compared to flatter regions where features are harder to predict.
- If not, consider analyzing the alpha weight in the loss function to see if the balance between L1 and SSIM is skewed, causing the network to rely too heavily on one component.

- [1] Fabio Merizzi, Andrea Asperti, and Stefano Colamonaco. "Wind speed super-resolution and validation: from ERA5 to CERRA via diffusion models". In: *Neural Computing and Applications* 36.34 (2024), pp. 21899–21921.
- [2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5–9, 2015, proceedings, part III* 18. Springer. 2015, pp. 234–241.
- [3] Robbie A Watt and Laura A Mansfield. "Generative diffusion-based downscaling for climate". In: *arXiv preprint arXiv:2404.17752* (2024).