BKN_top_MSE_with_terrain_grad

February 25, 2026

[1]:
```
from pathlib import Path
from scipy.io import loadmat
import sys
import os

# Robust path finding for data.mat
current_path = Path.cwd()
possible_data_paths = [
    current_path / 'data' / 'data.mat',
    current_path.parent / 'data' / 'data.mat',
    current_path.parent.parent / 'data' / 'data.mat',
    # Fallback absolute path
    Path('/home/luky/skola/KalmanNet-for-state-estimation/data/data.mat')
]

dataset_path = None
for p in possible_data_paths:
    if p.exists():
        dataset_path = p
        break

if dataset_path is None or not dataset_path.exists():
    print("Warning: data.mat not found automatically.")
    dataset_path = Path('data/data.mat')

print(f"Dataset path: {dataset_path}")

# Add project root to sys.path (2 levels up from debug/test)
notebook_dir = os.getcwd()
project_root = os.path.abspath(os.path.join(notebook_dir, '..', '..'))
if project_root not in sys.path:
    sys.path.insert(0, project_root)
print(f"Project root added: {project_root}")

mat_data = loadmat(dataset_path)
print(mat_data.keys())
```

```
Dataset path: /home/luky/skola/KalmanNet-main/data/data.mat
```

```
Project root added: /home/luky/skola/KalmanNet-main
dict_keys(['__header__', '__version__', '__globals__', 'hB', 'souradniceGNSS',
'souradniceX', 'souradniceY', 'souradniceZ'])
```

[2]:
```python
import torch
import matplotlib.pyplot as plt
from utils import trainer
from utils import utils
from Systems import DynamicSystem
import Filters
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
from scipy.io import loadmat
from scipy.interpolate import RegularGridInterpolator
import random

torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(42)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"device: {device}")
```

```
device: cuda
```

[3]:
```python
mat_data = loadmat(dataset_path)

souradniceX_mapa = mat_data['souradniceX']
souradniceY_mapa = mat_data['souradniceY']
souradniceZ_mapa = mat_data['souradniceZ']
souradniceGNSS = mat_data['souradniceGNSS']
x_axis_unique = souradniceX_mapa[0, :]
y_axis_unique = souradniceY_mapa[:, 0]

print(f"Dimensions of 1D X axis: {x_axis_unique.shape}")
print(f"Dimensions of 1D Y axis: {y_axis_unique.shape}")
print(f"Dimensions of 2D elevation data Z: {souradniceZ_mapa.shape}")

terMap_interpolator = RegularGridInterpolator(
    (y_axis_unique, x_axis_unique),
    souradniceZ_mapa,
    bounds_error=False,
    fill_value=np.nan
)
```

```
def terMap(px, py):
    # Query bilinear interpolation over the terrain map
    points_to_query = np.column_stack((py, px))
    return terMap_interpolator(points_to_query)
```

```
Dimensions of 1D X axis: (2500,)
Dimensions of 1D Y axis: (2500,)
Dimensions of 2D elevation data Z: (2500, 2500)
```

[4]:
```
import torch
from Systems import DynamicSystemTAN

state_dim = 4
obs_dim = 3
dT = 1
q = 1

F = torch.tensor([[1.0, 0.0, dT, 0.0],
                  [0.0, 1.0, 0.0, dT],
                  [0.0, 0.0, 1.0, 0.0],
                  [0.0, 0.0, 0.0, 1.0]])

Q = q* torch.tensor([[dT**3/3, 0.0, dT**2/2, 0.0],
                  [0.0, dT**3/3, 0.0, dT**2/2],
                  [dT**2/2, 0.0, dT, 0.0],
                  [0.0, dT**2/2, 0.0, dT]])
R = torch.tensor([[3.0**2, 0.0, 0.0],
                  [0.0, 1.0**2, 0.0],
                  [0.0, 0.0, 1.0**2]])

initial_velocity_np = souradniceGNSS[:2, 1] - souradniceGNSS[:2, 0]
# initial_velocity_np = torch.from_numpy()
initial_velocity = torch.from_numpy(np.array([0,0]))

initial_position = torch.from_numpy(souradniceGNSS[:2, 0])
x_0 = torch.cat([
    initial_position,
    initial_velocity
]).float()
print(x_0)

P_0 = torch.tensor([[25.0, 0.0, 0.0, 0.0],
                    [0.0, 25.0, 0.0, 0.0],
                    [0.0, 0.0, 0.5, 0.0],
                    [0.0, 0.0, 0.0, 0.5]])
import torch.nn.functional as func
```

```
def h_nl_differentiable(x: torch.Tensor, map_tensor, x_min, x_max, y_min,␣
 ↪y_max) -> torch.Tensor:
    batch_size = x.shape[0]

    px = x[:, 0]
    py = x[:, 1]

    px_norm = 2.0 * (px - x_min) / (x_max - x_min) - 1.0
    py_norm = 2.0 * (py - y_min) / (y_max - y_min) - 1.0

    sampling_grid = torch.stack((px_norm, py_norm), dim=1).view(batch_size, 1,␣
 ↪1, 2)

    vyska_terenu_batch = func.grid_sample(
        map_tensor.expand(batch_size, -1, -1, -1),
        sampling_grid,
        mode='bilinear',
        padding_mode='border',
        align_corners=True
    )

    vyska_terenu = vyska_terenu_batch.view(batch_size)

    eps = 1e-12
    vx_w, vy_w = x[:, 2], x[:, 3]
    norm_v_w = torch.sqrt(vx_w**2 + vy_w**2).clamp(min=eps)
    cos_psi = vx_w / norm_v_w
    sin_psi = vy_w / norm_v_w

    vx_b = cos_psi * vx_w - sin_psi * vy_w
    vy_b = sin_psi * vx_w + cos_psi * vy_w

    result = torch.stack([vyska_terenu, vx_b, vy_b], dim=1)

    return result

x_axis_unique = souradniceX_mapa[0, :]
y_axis_unique = souradniceY_mapa[:, 0]
terMap_tensor = torch.from_numpy(souradniceZ_mapa).float().unsqueeze(0).
 ↪unsqueeze(0).to(device)
x_min, x_max = x_axis_unique.min(), x_axis_unique.max()
y_min, y_max = y_axis_unique.min(), y_axis_unique.max()

h_wrapper = lambda x: h_nl_differentiable(
    x,
    map_tensor=terMap_tensor,
```

```
    x_min=x_min,
    x_max=x_max,
    y_min=y_min,
    y_max=y_max
)

system_model = DynamicSystemTAN(
    state_dim=state_dim,
    obs_dim=obs_dim,
    Q=Q.float(),
    R=R.float(),
    Ex0=x_0.float(),
    P0=P_0.float(),
    F=F.float(),
    h=h_wrapper,
    x_axis_unique=x_axis_unique,
    y_axis_unique=y_axis_unique,
    device=device
)
```

```
tensor([1487547.1250, 6395520.5000,       0.0000,       0.0000])
INFO: DynamicSystemTAN inicializován s hranicemi mapy:
  X: [1476611.42, 1489541.47]
  Y: [6384032.63, 6400441.34]
```

[5]:
```
import torch
from torch.utils.data import TensorDataset, DataLoader
from utils import utils
import torch
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import os
import random
from copy import deepcopy
from state_NN_models import TAN
from utils import trainer

torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
```

[6]:
```
import torch
from torch.utils.data import TensorDataset, DataLoader
import os
```

```python
from utils import trainer # Předpokládám, že toto máš

# === 1. ZJEDNODUŠENÝ DATA MANAGER (BEZ NORMALIZACE) ===
class NavigationDataManager:
    def __init__(self, data_dir):
        """
        Jen držák na cestu k datům. Žádná statistika, žádná normalizace.
        """
        self.data_dir = data_dir

    def get_dataloader(self, seq_len, split='train', shuffle=True,␣
 ↪batch_size=32):
        # Sestavení cesty: ./generated_data/len_100/train.pt
        path = os.path.join(self.data_dir, f'len_{seq_len}', f'{split}.pt')

        if not os.path.exists(path):
            raise FileNotFoundError(f" Dataset nenalezen: {path}")

        # Načtení tenzorů
        data = torch.load(path)
        x = data['x'] # Stav [Batch, Seq, DimX]
        y = data['y'] # Měření [Batch, Seq, DimY] - RAW DATA

        # Vytvoření datasetu
        dataset = TensorDataset(x, y)

        return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)

# === 2. KONFIGURACE CURRICULA ===
DATA_DIR = './generated_data_synthetic_controlled'

# Inicializace manažera (teď je to jen wrapper pro načítání souborů)
data_manager = NavigationDataManager(DATA_DIR)

# Definice fází (zde řídíš, jak se trénink vyvíjí)
curriculum_schedule = [
    # FÁZE 1: Warm-up (Krátké sekvence)
    {
        'phase_id': 1,
        'seq_len': 10,
        'epochs': 500,
        'lr': 1e-3,
        'batch_size': 256
    },

    # FÁZE 2: Stabilizace (Střední délka)
    {
```

```
        'phase_id': 2,
        'seq_len': 100,
        'epochs': 200,
        'lr': 1e-4,
        'batch_size': 64
    },

    # FÁZE 3: Long-term Reality (Plná délka)
    {
        'phase_id': 3,
        'seq_len': 300,
        'epochs': 200,
        'lr': 1e-5,
        'batch_size': 64       # Menší batch kvůli paměti GPU u dlouhých␣
 ↪sekvencí
    }
]

# === 3. NAČÍTÁNÍ DO PAMĚTI (CACHING) ===
print("\n=== NAČÍTÁNÍ RAW DAT Z DISKU (BEZ EXT. NORMALIZACE) ===")
datasets_cache = {}

for phase in curriculum_schedule:
    seq_len = phase['seq_len']
    bs = phase['batch_size']

    print(f"  Načítám Fázi {phase['phase_id']}: Seq={seq_len} | Batch={bs} ...")

    try:
        # Použití DataManageru
        train_loader = data_manager.get_dataloader(seq_len=seq_len,␣
 ↪split='train', shuffle=True, batch_size=bs)
        val_loader = data_manager.get_dataloader(seq_len=seq_len, split='val',␣
 ↪shuffle=False, batch_size=bs)

        # Uložení do cache
        datasets_cache[phase['phase_id']] = (train_loader, val_loader)

        # Rychlá kontrola pro jistotu
        x_ex, y_ex = next(iter(train_loader))
        if phase['phase_id'] == 1:
            print(f"     Ukázka RAW dat (y): {y_ex[0, 0, :].tolist()}")
            # Měl bys vidět velká čísla (např. 250.0) a malá (0.2), ne ~0.0

    except FileNotFoundError as e:
        print(f"     CHYBA: {e}")
        # raise e # Odkomentuj, pokud chceš, aby to spadlo při chybě
```

```
print("\n Data připravena. Normalizaci řeší model.")
```

```
=== NAČÍTÁNÍ RAW DAT Z DISKU (BEZ EXT. NORMALIZACE) ===
 Načítám Fázi 1: Seq=10 | Batch=256 …
    Ukázka RAW dat (y): [323.7707824707031, -13.519903182983398,
-29.721908569335938]
 Načítám Fázi 2: Seq=100 | Batch=64 …
 Načítám Fázi 3: Seq=300 | Batch=64 …

 Data připravena. Normalizaci řeší model.
```

[7]:
```
import torch
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    """
    Bezpečná NLL loss funkce.
    """
    safe_var = var + min_var
    error_sq = (preds - target) ** 2
    # Clampujeme velikost chyby v čitateli, aby loss neexplodovala,
    # ale gradient do variance (ve jmenovateli) zůstal zachován.
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)
    return nll.mean()

def train_BayesianKalmanNet_TwoPhase(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad,
    J_samples, validation_period, logging_period,
    mse_warmup_iters=0,  # <--- NOVÝ PARAMETR: Kolik iterací trénovat jen na MSE
    weight_decay_=1e-5
):
    # Optimizer
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay_)

    # Scheduler (volitelný, zde vypnutý pro jednoduchost)
    # scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,␣
 ↪mode='min', factor=0.5, patience=50)

    best_val_metric = float('inf') # Buď MSE nebo ANEES podle fáze
    best_model_state = None
```

```
    best_iter_count = 0
    train_iter_count = 0
    done = False

    print(f"  START Two-Phase Training")
    print(f"    Phase 1: MSE Warmup (0 - {mse_warmup_iters} iters)")
    print(f"    Phase 2: NLL Optimization ({mse_warmup_iters} -␣
↪{total_train_iter} iters)")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            # Detekce NaN v datech
            if torch.isnan(x_true_batch).any():
                print(f"!!! SKIP BATCH iter {train_iter_count}: NaN found in␣
↪x_true !!!")
                continue

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)
            batch_size, seq_len, _ = x_true_batch.shape

            # --- Training Step ---
            optimizer.zero_grad()

            all_trajectories_for_ensemble = []
            all_regs_for_ensemble = []

            # 1. Ensemble Forward Pass
            for j in range(J_samples):
                model.reset(batch_size=batch_size, initial_state=x_true_batch[:
↪, 0, :])
                current_trajectory_x_hats = []
                current_trajectory_regs = []

                for t in range(1, seq_len):
                    y_t = y_meas_batch[:, t, :]
                    x_filtered_t, reg_t = model.step(y_t)

                    if torch.isnan(x_filtered_t).any():
                        raise ValueError(f"NaN in x_filtered_t at sample {j},␣
↪step {t}")

                    current_trajectory_x_hats.append(x_filtered_t)
                    current_trajectory_regs.append(reg_t)
```

```
                all_trajectories_for_ensemble.append(torch.
↪stack(current_trajectory_x_hats, dim=1))
                all_regs_for_ensemble.append(torch.sum(torch.
↪stack(current_trajectory_regs)))

            # 2. Statistiky Ensemble
            ensemble_trajectories = torch.stack(all_trajectories_for_ensemble,␣
↪dim=0)
            x_hat_sequence = ensemble_trajectories.mean(dim=0)

            # Epistemická variance
            cov_diag_sequence = ensemble_trajectories.var(dim=0) + 1e-9

            # Normalizovaná regularizace (na délku sekvence)
            regularization_loss = torch.stack(all_regs_for_ensemble).mean() /␣
↪seq_len

            target_sequence = x_true_batch[:, 1:, :]

            # --- 3. VÝPOČET LOSS (Dvě fáze) ---

            # Vždy spočítáme obojí pro logování
            mse_loss = F.mse_loss(x_hat_sequence, target_sequence)
            nll_loss = gaussian_nll_safe(
                target=target_sequence,
                preds=x_hat_sequence,
                var=cov_diag_sequence,
                min_var=1e-5,
                max_error_sq=100.0
            )

            # Rozhodování o optimalizační Loss
            loss_mode = ""
            if train_iter_count < mse_warmup_iters:
                # FÁZE 1: Warmup na MSE
                # Ignorujeme NLL, soustředíme se na trefení trajektorie
                loss = mse_loss + regularization_loss
                loss_mode = "MSE_WARMUP"
            else:
                # FÁZE 2: NLL Optimalizace
                # Minimalizujeme NLL (kalibrace nejistoty + přesnost)
                # MSE zde není přímo v gradientu (je schované v NLL), ale␣
↪logujeme ho
                loss = nll_loss + regularization_loss
                loss_mode = "NLL_OPTIM"
```

```
            if torch.isnan(loss):
                print("Collapse detected (NaN loss)"); done = True; break

            loss.backward()

            # --- DIAGNOSTIKA GRADIENTŮ ---
            if train_iter_count % logging_period == 0:
                total_norm = 0.0
                max_grad = 0.0
                nan_grad_detected = False
                for p in model.parameters():
                    if p.grad is not None:
                        param_norm = p.grad.data.norm(2).item()
                        total_norm += param_norm ** 2
                        p_max = p.grad.data.abs().max().item()
                        if p_max > max_grad: max_grad = p_max
                        if torch.isnan(p.grad).any():
                            nan_grad_detected = True
                total_norm = total_norm ** 0.5

                if nan_grad_detected:
                     print(f"!!! WARNING: NaN gradient detected at iter␣
↪{train_iter_count} !!!")

            if clip_grad > 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(), clip_grad)

            optimizer.step()
            train_iter_count += 1

            # --- LOGGING ---
            if train_iter_count % logging_period == 0:
                with torch.no_grad():
                    # Statistiky variance
                    min_variance = cov_diag_sequence.min().item()
                    max_variance = cov_diag_sequence.max().item()
                    mean_variance = cov_diag_sequence.mean().item()

                    # Dropout pravděpodobnosti
                    p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                    p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                    # Chyba v metrech (L1)
                    diff = x_hat_sequence - target_sequence
                    mean_error = diff.abs().mean().item()
```

```
                print(f"--- Iter [{train_iter_count}/{total_train_iter}]␣
↪({loss_mode}) ---")
                print(f"    Total Loss:     {loss.item():.4f}")
                print(f"    MSE (Metric):   {mse_loss.item():.4f}")
                print(f"    NLL (Metric):   {nll_loss.item():.4f}")
                print(f"    Reg Loss:       {regularization_loss.item():.6f}")
                print(f"    Var Stats:      Min={min_variance:.2e},␣
↪Max={max_variance:.2e}, Mean={mean_variance:.2e}")
                print(f"    Mean Error L1:  {mean_error:.4f} m")
                print(f"    Grad Norm:      {total_norm:.4f} (Max: {max_grad:.
↪4f})")
                print(f"    Dropout:        p1={p1:.4f}, p2={p2:.4f}")

# --- VALIDATION (Upravená část) ---
            if train_iter_count > 0 and train_iter_count % validation_period ==␣
↪0:
                print(f"\n--- Validation at iteration {train_iter_count} ---")
                model.eval()
                val_mse_list = []

                # Pro ANEES
                all_val_x_true, all_val_x_hat, all_val_P_hat = [], [], []

                with torch.no_grad():
                    for x_true_val, y_meas_val in val_loader:
                        v_bs, v_seq, _ = x_true_val.shape
                        x_true_val = x_true_val.to(device)
                        y_meas_val = y_meas_val.to(device)

                        val_ensemble_trajs = []

                        for j in range(J_samples):
                            model.reset(batch_size=v_bs,␣
↪initial_state=x_true_val[:, 0, :])
                            v_x_hats = []
                            for t in range(1, v_seq):
                                est, _ = model.step(y_meas_val[:, t, :])
                                v_x_hats.append(est)
                            val_ensemble_trajs.append(torch.stack(v_x_hats,␣
↪dim=1))

                        val_ens_stack = torch.stack(val_ensemble_trajs, dim=0)␣
↪# [J, B, T, D]
                        val_mean = val_ens_stack.mean(dim=0)
                        val_var_diag = val_ens_stack.var(dim=0) + 1e-9
```

```
                        val_mse_list.append(F.mse_loss(val_mean, x_true_val[:,␣
↪1:, :]).item())

                        # --- Příprava dat pro ANEES ---
                        # 1. Stavy (Ground Truth a Odhad)
                        # Přidáme startovní bod (t=0), abychom měli celou␣
↪sekvenci
                        full_x_hat = torch.cat([x_true_val[:, 0, :].
↪unsqueeze(1), val_mean], dim=1)

                        # 2. Kovariance (P)
                        # Vytvoříme plné matice 4x4 z diagonálního rozptylu
                        val_covs_full = torch.zeros(v_bs, v_seq-1, 4, 4,␣
↪device=device)
                        # Rychlá vektorizovaná konstrukce diagonály
                        # (Místo cyklu přes batch a čas použijeme diag_embed,␣
↪pokud to PyTorch verze umí)
                        try:
                            val_covs_full = torch.diag_embed(val_var_diag)
                        except:
                            # Fallback pro starší verze nebo pokud to selže
                            for b in range(v_bs):
                                for t in range(v_seq-1):
                                    val_covs_full[b, t] = torch.
↪diag(val_var_diag[b, t])

                        # P0 (počáteční nejistota - malá)
                        P0 = torch.eye(4, device=device).unsqueeze(0).
↪unsqueeze(0).repeat(v_bs, 1, 1, 1) * 1e-6
                        full_P_hat = torch.cat([P0, val_covs_full], dim=1)

                        all_val_x_true.append(x_true_val.cpu())
                        all_val_x_hat.append(full_x_hat.cpu())
                        all_val_P_hat.append(full_P_hat.cpu())

                avg_val_mse = np.mean(val_mse_list)

                # --- VÝPOČET ANEES ---
                try:
                    cat_true = torch.cat(all_val_x_true, dim=0)
                    cat_hat = torch.cat(all_val_x_hat, dim=0)
                    cat_P = torch.cat(all_val_P_hat, dim=0)

                    # Voláme vaši funkci (předpokládám, že je v modulu␣
↪'trainer' nebo 'utils')
```

```
                    # Pokud ji nemáte naimportovanou, musíte ji definovat.
                    # Zde používám 'trainer.calculate_anees_vectorized' z␣
↪vašeho původního kódu
                    if hasattr(trainer, 'calculate_anees_vectorized'):
                        avg_val_anees = trainer.
↪calculate_anees_vectorized(cat_true, cat_hat, cat_P)
                    else:
                        avg_val_anees = float('nan') # Placeholder

                except Exception as e:
                    print(f"Error calculating ANEES: {e}")
                    avg_val_anees = float('nan')

                print(f"  Avg MSE: {avg_val_mse:.4f} | Avg ANEES:␣
↪{avg_val_anees:.4f}")

                # --- LOGIKA UKLÁDÁNÍ (Smart Saving) ---
                # Ukládáme, pokud se zlepší MSE (priorita 1).
                # Volitelně: Ve fázi NLL by se dalo ukládat, pokud se zlepší␣
↪ANEES (k ideální 4.0),
                # ale to je riskantní, pokud by MSE vyletělo.
                # Zůstaňme u MSE jako "kotvy kvality".

                current_metric = avg_val_mse
                if current_metric < best_val_metric:
                    print(f"  >>> New Best Model! (MSE: {best_val_metric:.4f}␣
↪-> {current_metric:.4f}) <<<")
                    best_val_metric = current_metric
                    best_iter_count = train_iter_count
                    best_model_state = deepcopy(model.state_dict())

                print("-" * 50)
                model.train()

    print("\nTraining completed.")
    if best_model_state:
        print(f"Loading best model from iteration {best_iter_count}")
        model.load_state_dict(best_model_state)

    return {"final_model": model}
```

[8]:
```
import torch
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

def calculate_anees_vectorized(x_true, x_hat, P_hat, eps=1e-6):
```

```
    """
    Robustní výpočet ANEES pomocí Pseudo-Inverze.
    Zvládá i situaci, kdy J_samples < State_Dim (singulární matice).
    """
    # 1. Zploštění (Flattening)
    if x_true.dim() == 3: x_true = x_true.reshape(-1, x_true.shape[-1])
    if x_hat.dim() == 3: x_hat = x_hat.reshape(-1, x_hat.shape[-1])
    if P_hat.dim() == 4: P_hat = P_hat.reshape(-1, P_hat.shape[-2], P_hat.
↪shape[-1])

    # 2. Výpočet chyby
    error = (x_true - x_hat).unsqueeze(-1)  # [N, Dim, 1]

    # 3. Pseudo-Inverze matice P
    # pinv je mnohem stabilnější pro Low-Rank matice (když máme málo vzorků)
    # hermitian=True říká, že matice je symetrická (což kovariance je)
    P_inv = torch.linalg.pinv(P_hat, hermitian=True)

    # 4. ANEES = error^T * P_inv * error
    # bmm: [N, 1, Dim] @ [N, Dim, Dim] -> [N, 1, Dim]
    #      [N, 1, Dim] @ [N, Dim, 1] -> [N, 1, 1]

    temp = torch.bmm(error.transpose(1, 2), P_inv)
    anees_per_sample = torch.bmm(temp, error).squeeze() # [N]

    # Ochrana proti numerickým artefaktům (malinká záporná čísla jako -1e-10)
    anees_per_sample = torch.clamp(anees_per_sample, min=0.0)

    return anees_per_sample.mean().item()

def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    # 1. Bezpečná variance (epsilon) - správně
    safe_var = var + min_var

    # 2. Kvadratická chyba
    error_sq = (preds - target) ** 2

    # 3. === OPRAVA ===
    # Clampujeme "velikost chyby", nikoliv "velikost trestu".
    # Tím říkáme: "Pokud je chyba větší než 10m (100m^2), chovej se, jako by␣
↪byla 10m."
    # Vzorec zůstává: Const / var.
    # Derivace je: -Const / var^2. (To je záporné číslo -> zvyšování var␣
↪snižuje Loss -> SPRÁVNĚ!)
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)

    # 4. Výpočet s oříznutou chybou
```

```
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)

    return nll.mean()

def train_BayesianKalmanNet_Hybrid(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad,
    J_samples, validation_period, logging_period,
    warmup_iterations=0, weight_decay_=1e-5,
    lambda_mse=100.0  # <--- NOVÝ PARAMETR: Kotva pro MSE
):
    # torch.autograd.set_detect_anomaly(True)
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay_)

    # Scheduler: Pokud se loss zasekne, snížíme LR (pomáhá stabilizovat␣
 ↪konvergenci)
    # scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    #     optimizer, mode='min', factor=0.5, patience=50
    # )

    best_val_anees = float('inf')
    score_at_best = {"val_nll": 0.0, "val_mse": 0.0}
    best_iter_count = 0
    best_model_state = None
    train_iter_count = 0
    done = False

    print(f"  START Hybrid Training: Loss = NLL + {lambda_mse} * MSE")
    print(f"    Logging period: {logging_period} iterations")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break
            if torch.isnan(x_true_batch).any():
                print(f"!!! SKIP BATCH iter {train_iter_count}: NaN found in␣
 ↪x_true (Ground Truth) !!!")
                continue

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)

            # --- Training ---
            optimizer.zero_grad()
            batch_size, seq_len, _ = x_true_batch.shape
```

```
            all_trajectories_for_ensemble = []
            all_regs_for_ensemble = []

            # 1. Ensemble Forward Pass
            for j in range(J_samples):
                model.reset(batch_size=batch_size, initial_state=x_true_batch[:
↪, 0, :])
                current_trajectory_x_hats = []
                current_trajectory_regs = []
                for t in range(1, seq_len):
                    y_t = y_meas_batch[:, t, :]
                    x_filtered_t, reg_t = model.step(y_t)
                    if torch.isnan(x_filtered_t).any():
                            raise ValueError(f"NaN in x_filtered_t at sample␣
↪{j}, step {t}")
                    current_trajectory_x_hats.append(x_filtered_t)
                    current_trajectory_regs.append(reg_t)
                all_trajectories_for_ensemble.append(torch.
↪stack(current_trajectory_x_hats, dim=1))
                all_regs_for_ensemble.append(torch.sum(torch.
↪stack(current_trajectory_regs)))

            # 2. Statistiky Ensemble
            ensemble_trajectories = torch.stack(all_trajectories_for_ensemble,␣
↪dim=0)
            x_hat_sequence = ensemble_trajectories.mean(dim=0)

            # Epistemická variance (čistý rozptyl sítě)
            # Přičítáme 1e-9 jen proti dělení nulou, není to "noise floor"
            cov_diag_sequence = ensemble_trajectories.var(dim=0) + 1e-9

            regularization_loss = torch.stack(all_regs_for_ensemble).mean()/
↪seq_len
            target_sequence = x_true_batch[:, 1:, :]

            # --- 3. VÝPOČET HYBRIDNÍ LOSS ---

            # A) MSE Část (Přesnost)
            mse_loss = F.mse_loss(x_hat_sequence, target_sequence)

            # B) NLL Část (Konzistence)
            # 0.5 * (log(var) + (target - pred)^2 / var)
            cov_diag_clamped = torch.clamp(cov_diag_sequence, min=1e-4, max=1e6)
            error_sq = (x_hat_sequence - target_sequence) ** 2
            nll_term = 0.5 * (torch.log(cov_diag_clamped) + error_sq /␣
↪cov_diag_clamped)
```

```
            nll_loss = gaussian_nll_safe(
                target=target_sequence,
                preds=x_hat_sequence,
                var=cov_diag_sequence,
                min_var=1e-5,       # Epsilon pro stabilitu
                max_error_sq=100.0 # Ořezání extrémních chyb (pokud je␣
↪implementováno)
            )
            mean_var = cov_diag_sequence.mean()
            var_penalty = torch.relu(mean_var - 100.0) * 0.01

            # C) Celková Loss (Hybrid)
            # Zde je ta magie: I když NLL chce utéct s variancí, lambda_mse *␣
↪mse ho drží zpátky
            weighted_mse = lambda_mse * mse_loss
            loss = nll_loss + weighted_mse + regularization_loss * 10.0 +␣
↪var_penalty

            if torch.isnan(loss):
                print("Collapse detected (NaN loss)"); done = True; break

            loss.backward()

            # --- DIAGNOSTIC LOGGING (Gradients) ---
            # Zaznamenáme statistiky gradientů před oříznutím (clippingem)
            if train_iter_count % logging_period == 0:
                total_norm = 0.0
                max_grad = 0.0
                min_grad = float('inf')
                nan_grad_detected = False
                for p in model.parameters():
                    if p.grad is not None:
                        param_norm = p.grad.data.norm(2).item()
                        total_norm += param_norm ** 2
                        p_max = p.grad.data.abs().max().item()
                        p_min = p.grad.data.abs().min().item()
                        if p_max > max_grad: max_grad = p_max
                        if p_min < min_grad: min_grad = p_min
                        if torch.isnan(p.grad).any():
                            nan_grad_detected = True
                total_norm = total_norm ** 0.5

                if nan_grad_detected:
                     print(f"!!! WARNING: NaN gradient detected at iter␣
↪{train_iter_count} !!!")
```

```
            if clip_grad > 0: torch.nn.utils.clip_grad_norm_(model.
↪parameters(), clip_grad)
            optimizer.step()
            train_iter_count += 1

            # --- Logging ---
            diff = x_hat_sequence - target_sequence
            mean_error = diff.abs().mean().item()
            min_variance = cov_diag_sequence.min().item()
            max_variance = cov_diag_sequence.max().item()
            mean_variance = cov_diag_sequence.mean().item()

            if train_iter_count % logging_period == 0:
                with torch.no_grad():
                    # Zjistíme dropout pravděpodobnosti (jen pro info)
                    p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                    p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                print(f"--- Iter [{train_iter_count}/{total_train_iter}] ---")
                print(f"    Total Loss:     {loss.item():.4f}")
                print(f"    MSE (Raw):      {mse_loss.item():.6f}")
                print(f"    MSE (Weighted): {weighted_mse.item():.4f}␣
↪(lambda={lambda_mse})")
                print(f"    NLL Component:  {nll_loss.item():.4f}")
                print(f"    Reg Loss:       {regularization_loss.item():.6f}")
                print(f"    Variance stats: Min={min_variance:.2e},␣
↪Max={max_variance:.2e}, Mean={mean_variance:.2e}")
                print(f"    Mean Error L1:  {mean_error:.4f}")
                print(f"    Grad Norm:      {total_norm:.4f} (Max abs grad:␣
↪{max_grad:.4f})")
                print(f"    Dropout probs:  p1={p1:.4f}, p2={p2:.4f}")

                # Check pro "Variance collapse"
                if mean_variance < 1e-8:
                    print("    !!! WARNING: Variance is extremely low (Collapse␣
↪risk) !!!")

            # --- Validation step ---
            if train_iter_count > 0 and train_iter_count % validation_period ==␣
↪0:
                # Step scheduleru podle trénovací loss (nebo validace, pokud␣
↪bys to předělal)
                # scheduler.step(loss)
```

```
                print(f"\n--- Validation at iteration {train_iter_count} ---")
                model.eval()
                val_mse_list = []
                all_val_x_true_cpu, all_val_x_hat_cpu, all_val_P_hat_cpu = [],␣
↪[], []

                with torch.no_grad():
                    for x_true_val_batch, y_meas_val_batch in val_loader:
                        val_batch_size, val_seq_len, _ = x_true_val_batch.shape
                        x_true_val_batch = x_true_val_batch.to(device)
                        y_meas_val_batch = y_meas_val_batch.to(device)
                        val_ensemble_trajectories = []
                        for j in range(J_samples):
                            model.reset(batch_size=val_batch_size,␣
↪initial_state=x_true_val_batch[:, 0, :])
                            val_current_x_hats = []
                            for t in range(1, val_seq_len):
                                y_t_val = y_meas_val_batch[:, t, :]
                                x_filtered_t, _ = model.step(y_t_val)
                                val_current_x_hats.append(x_filtered_t)
                            val_ensemble_trajectories.append(torch.
↪stack(val_current_x_hats, dim=1))

                        # Agregace validace
                        val_ensemble = torch.stack(val_ensemble_trajectories,␣
↪dim=0)
                        val_preds_seq = val_ensemble.mean(dim=0)

                        val_target_seq = x_true_val_batch[:, 1:, :]
                        val_mse_list.append(F.mse_loss(val_preds_seq,␣
↪val_target_seq).item())

                        # Příprava pro ANEES
                        initial_state_val = x_true_val_batch[:, 0, :].
↪unsqueeze(1)
                        full_x_hat = torch.cat([initial_state_val,␣
↪val_preds_seq], dim=1)

                        # Epistemická variance
                        val_covs_diag = val_ensemble.var(dim=0) + 1e-9

                        # Vytvoření diagonálních matic P
                        # (Zjednodušená konstrukce pro ANEES calc)
                        # Pro přesné ANEES bychom měli dělat outer product,
                        # ale diagonála z var() je dobrá aproximace pro BKN
```

```
                        val_covs_full = torch.zeros(val_batch_size,␣
↪val_seq_len-1, 4, 4, device=device)
                        for b in range(val_batch_size):
                            for t in range(val_seq_len-1):
                                val_covs_full[b, t] = torch.
↪diag(val_covs_diag[b, t])

                        P0 = model.system_model.P0.unsqueeze(0).
↪repeat(val_batch_size, 1, 1).unsqueeze(1)
                        full_P_hat = torch.cat([P0, val_covs_full], dim=1)

                        all_val_x_true_cpu.append(x_true_val_batch.cpu())
                        all_val_x_hat_cpu.append(full_x_hat.cpu())
                        all_val_P_hat_cpu.append(full_P_hat.cpu())

                avg_val_mse = np.mean(val_mse_list)
                final_x_true_list = torch.cat(all_val_x_true_cpu, dim=0)
                final_x_hat_list = torch.cat(all_val_x_hat_cpu, dim=0)
                final_P_hat_list = torch.cat(all_val_P_hat_cpu, dim=0)

                # Výpočet ANEES
                try:
                    avg_val_anees =␣
↪calculate_anees_vectorized(final_x_true_list, final_x_hat_list,␣
↪final_P_hat_list)
                except Exception as e:
                    print(f"  !!! Error calculating ANEES: {e}")
                    avg_val_anees = float('nan')

                print(f"  Average MSE: {avg_val_mse:.4f}, Average ANEES:␣
↪{avg_val_anees:.4f}")

                # Ukládání modelu:
                if not np.isnan(avg_val_anees) and avg_val_anees <␣
↪best_val_anees and avg_val_anees > 0:
                    print(f"  >>> New best VALIDATION ANEES! Saving model. (Old:
↪ {best_val_anees:.4f} -> New: {avg_val_anees:.4f}) <<<")
                    best_val_anees = avg_val_anees
                    best_iter_count = train_iter_count
                    score_at_best['val_mse'] = avg_val_mse
                    best_model_state = deepcopy(model.state_dict())
                print("-" * 50)
                model.train()

    print("\nTraining completed.")
    if best_model_state:
```

```
        print(f"Loading best model from iteration {best_iter_count} with ANEES␣
 ↪{best_val_anees:.4f}")
        model.load_state_dict(best_model_state)
    else:
        print("No best model was saved; returning last state.")

    return {
        "best_val_anees": best_val_anees,
        "best_val_nll": score_at_best['val_nll'],
        "best_val_mse": score_at_best['val_mse'],
        "best_iter": best_iter_count,
        "final_model": model
    }
```

[9]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

# Předpokládáme existenci helperu (pokud ne, definujte ho jako v minulé␣
 ↪odpovědi)
def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    safe_var = var + min_var
    error_sq = (preds - target) ** 2
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)
    return nll.mean()

def train_BayesianKalmanNet_TBPTT_TwoPhase(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad=1.0,
    J_samples=5, tbptt_steps=20,
    validation_period=50, logging_period=10,
    mse_warmup_iters=0, # Počet updatů pro MSE fázi
    weight_decay_=1e-5
):
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay_)

    best_val_mse = float('inf')
    best_model_state = None
    train_iter_count = 0 # Počítadlo updatů (gradient steps)
    done = False

    print(f"  START TBPTT Two-Phase Training (Window={tbptt_steps})")
    print(f"    Phase 1: MSE Warmup (0 - {mse_warmup_iters} steps)")
```

```
    print(f"    Phase 2: NLL Optimization ({mse_warmup_iters} -␣
↪{total_train_iter} steps)")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)

            batch_size, seq_len, dim_x = x_true_batch.shape

            # --- SUPER BATCH (Vektorizace) ---
            x_true_super = x_true_batch.repeat_interleave(J_samples, dim=0)
            y_meas_super = y_meas_batch.repeat_interleave(J_samples, dim=0)
            super_batch_size = x_true_super.shape[0]

            # 1. Reset na začátku sekvence
            model.reset(batch_size=super_batch_size,␣
↪initial_state=x_true_super[:, 0, :])

            # Gradienty nulujeme před sekvencí
            optimizer.zero_grad()

            # 2. TBPTT Smyčka přes okna
            for t_start in range(1, seq_len, tbptt_steps):
                if train_iter_count >= total_train_iter: done = True; break

                t_end = min(t_start + tbptt_steps, seq_len)
                current_window_len = t_end - t_start
                if current_window_len <= 0: continue

                # A) Forward pass oknem
                window_x_preds = []
                window_regs = []

                for t in range(t_start, t_end):
                    y_t = y_meas_super[:, t, :]
                    x_est, reg = model.step(y_t)
                    window_x_preds.append(x_est)
                    window_regs.append(reg)

                # B) Zpracování výsledků okna
                preds_super = torch.stack(window_x_preds, dim=1) # [Batch*J,␣
↪Window, 4]
                regs_super = torch.stack(window_regs)
```

```
                # Reshape pro statistiku [Batch, J, Window, 4]
                preds_reshaped = preds_super.view(batch_size, J_samples,␣
↪current_window_len, dim_x)

                x_hat_seq = preds_reshaped.mean(dim=1)
                cov_diag_seq = preds_reshaped.var(dim=1) + 1e-9

                target_seq = x_true_batch[:, t_start:t_end, :]

                # C) Výpočet Loss (Two-Phase)
                mse_loss = F.mse_loss(x_hat_seq, target_seq)

                # NLL (použijeme bezpečnou funkci)
                nll_loss = gaussian_nll_safe(target_seq, x_hat_seq,␣
↪cov_diag_seq, max_error_sq=100.0)

                # Regularizace (průměr na krok)
                reg_loss = regs_super.mean()

                # Rozhodování o Loss
                if train_iter_count < mse_warmup_iters:
                    loss = mse_loss + reg_loss
                    mode = "MSE"
                else:
                    # NLL Fáze (můžeme přidat malou kotvu MSE, např. 0.1, pro␣
↪stabilitu)
                    loss = nll_loss + reg_loss + (0.1 * mse_loss)
                    mode = "NLL"

                # D) Backward & Update
                loss.backward()

                if clip_grad > 0:
                    torch.nn.utils.clip_grad_norm_(model.parameters(),␣
↪clip_grad)

                optimizer.step()
                optimizer.zero_grad() # Důležité: nulujeme po každém kroku TBPTT

                # E) Detach Hidden State (Klíčové pro TBPTT)
                model.detach_hidden()

                train_iter_count += 1

                # --- Logging ---
                if train_iter_count % logging_period == 0:
```

```
                    with torch.no_grad():
                        p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                        p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()
                        diff = x_hat_seq - target_seq
                        mae = diff.abs().mean().item()

                    print(f"Iter {train_iter_count} ({mode}): Loss {loss.item():
↪.4f} | MSE {mse_loss.item():.2f} | NLL {nll_loss.item():.2f} | MAE {mae:.
↪2f}m")
                    print(f"    Dropout: p1={p1:.3f}, p2={p2:.3f} | VarMean:␣
↪{cov_diag_seq.mean().item():.1f}")

                # --- Validation (Uvnitř TBPTT smyčky) ---
                if train_iter_count % validation_period == 0:
                    model.eval()
                    val_mse_list = []

                    # Pro ANEES sběr
                    all_val_x_true, all_val_x_hat, all_val_P = [], [], []

                    with torch.no_grad():
                        for x_v, y_v in val_loader:
                            x_v, y_v = x_v.to(device), y_v.to(device)
                            b_v, s_v, _ = x_v.shape

                            # Validace běží Open-Loop na celé sekvenci (bez␣
↪TBPTT)
                            x_v_sup = x_v.repeat_interleave(J_samples, dim=0)
                            y_v_sup = y_v.repeat_interleave(J_samples, dim=0)

                            model.reset(batch_size=b_v*J_samples,␣
↪initial_state=x_v_sup[:,0,:])
                            preds_list = []

                            for ti in range(1, s_v):
                                est, _ = model.step(y_v_sup[:, ti, :])
                                preds_list.append(est)

                            preds_stack = torch.stack(preds_list, dim=1).
↪view(b_v, J_samples, s_v-1, 4)
                            val_mean = preds_stack.mean(dim=1)
                            val_var = preds_stack.var(dim=1) + 1e-9
```

```
                            val_mse_list.append(F.mse_loss(val_mean, x_v[:, 1:,␣
↪:]).item())

                            # Data pro ANEES
                            full_hat = torch.cat([x_v[:,0,:].unsqueeze(1),␣
↪val_mean], dim=1)
                            # Diagonal P construction
                            val_P_full = torch.zeros(b_v, s_v-1, 4, 4,␣
↪device=device)
                            for b in range(b_v):
                                for t in range(s_v-1):
                                    val_P_full[b,t] = torch.diag(val_var[b,t])
                            P0 = torch.eye(4, device=device).unsqueeze(0).
↪unsqueeze(0).repeat(b_v, 1, 1, 1)*1e-6
                            full_P = torch.cat([P0, val_P_full], dim=1)

                            all_val_x_true.append(x_v.cpu())
                            all_val_x_hat.append(full_hat.cpu())
                            all_val_P.append(full_P.cpu())

                    avg_val_mse = np.mean(val_mse_list)

                    # Calc ANEES
                    try:
                        cat_true = torch.cat(all_val_x_true, dim=0)
                        cat_hat = torch.cat(all_val_x_hat, dim=0)
                        cat_P = torch.cat(all_val_P, dim=0)
                        # Pokud máte funkci importovanou
                        avg_anees = calculate_anees_vectorized(cat_true,␣
↪cat_hat, cat_P)
                    except:
                        avg_anees = float('nan')

                    print(f"\n--- VALIDATION: MSE {avg_val_mse:.2f} | ANEES␣
↪{avg_anees:.2f} ---")

                    # Ukládání (podle MSE)
                    if avg_val_mse < best_val_mse:
                        print(f"  >>> New Best Model! (Old: {best_val_mse:.2f}␣
↪-> New: {avg_val_mse:.2f}) <<<")
                        best_val_mse = avg_val_mse
                        best_model_state = deepcopy(model.state_dict())
                    print("-" * 40)

                    model.train()
```

```
    print("TBPTT Training completed.")
    if best_model_state:
        model.load_state_dict(best_model_state)
    return {"final_model": model}
```

[10]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    safe_var = var + min_var
    error_sq = (preds - target) ** 2
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)
    return nll.mean()
def calculate_anees_vectorized(x_true, x_hat, P_hat, eps=1e-6):
    """
    Robustní výpočet ANEES pomocí Pseudo-Inverze.
    Zvládá i situaci, kdy J_samples < State_Dim (singulární matice).
    """
    # 1. Zploštění (Flattening)
    if x_true.dim() == 3: x_true = x_true.reshape(-1, x_true.shape[-1])
    if x_hat.dim() == 3: x_hat = x_hat.reshape(-1, x_hat.shape[-1])
    if P_hat.dim() == 4: P_hat = P_hat.reshape(-1, P_hat.shape[-2], P_hat.
 ↪shape[-1])

    # 2. Výpočet chyby
    error = (x_true - x_hat).unsqueeze(-1)  # [N, Dim, 1]

    # 3. Pseudo-Inverze matice P
    # pinv je mnohem stabilnější pro Low-Rank matice (když máme málo vzorků)
    # hermitian=True říká, že matice je symetrická (což kovariance je)
    P_inv = torch.linalg.pinv(P_hat, hermitian=True)

    # 4. ANEES = error^T * P_inv * error
    # bmm: [N, 1, Dim] @ [N, Dim, Dim] -> [N, 1, Dim]
    #      [N, 1, Dim] @ [N, Dim, 1] -> [N, 1, 1]

    temp = torch.bmm(error.transpose(1, 2), P_inv)
    anees_per_sample = torch.bmm(temp, error).squeeze() # [N]

    # Ochrana proti numerickým artefaktům (malinká záporná čísla jako -1e-10)
    anees_per_sample = torch.clamp(anees_per_sample, min=0.0)

    return anees_per_sample.mean().item()
```

```
def train_BayesianKalmanNet_TBPTT_Windowed(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad=1.0,
    J_samples=5,
    tbptt_k=5,
    tbptt_w=20,
    validation_period=50, logging_period=10,
    mse_warmup_iters=0,
    weight_decay_=1e-5,
    lambda_mse=100.0,
    calibration_parameter=10.0
):
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay_)

    best_val_score = float('inf')
    best_model_state = None
    train_iter_count = 0
    done = False

    if not hasattr(model, 'detach_hidden'):
         raise AttributeError("Modelu chybí metoda 'detach_hidden()'.")

    print(f"  START TBPTT Windowed Training (k={tbptt_k}, w={tbptt_w})")
    print(f"    Phase 1: MSE Warmup (0 - {mse_warmup_iters} updates)")
    print(f"    Phase 2: NLL Optimization (> {mse_warmup_iters} updates)")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)

            batch_size, seq_len, dim_x = x_true_batch.shape

            x_true_super = x_true_batch.repeat_interleave(J_samples, dim=0)
            y_meas_super = y_meas_batch.repeat_interleave(J_samples, dim=0)
            super_batch_size = x_true_super.shape[0]

            # 1. Hard Reset na začátku sekvence
            # Vyčistíme vše, aby reset vytvořil správné velikosti
            if hasattr(model, 'h_prev'): model.h_prev = None
            if hasattr(model, 'x_filtered_t_minus_1'): model.
 ↪x_filtered_t_minus_1 = None
```

```
            if hasattr(model, 'x_filtered_t_minus_2'): model.
↪x_filtered_t_minus_2 = None

            model.reset(batch_size=super_batch_size,␣
↪initial_state=x_true_super[:, 0, :])

            # 2. TBPTT Loop
            for t_start in range(1, seq_len, tbptt_w):
                if train_iter_count >= total_train_iter: done = True; break

                t_end = min(t_start + tbptt_w, seq_len)
                current_window_len = t_end - t_start
                if current_window_len <= 0: continue

                optimizer.zero_grad()
                window_x_preds = []
                window_regs = []

                # A) Forward pass
                for t in range(t_start, t_end):
                    y_t = y_meas_super[:, t, :]
                    x_est, reg = model.step(y_t)
                    window_x_preds.append(x_est)
                    window_regs.append(reg)

                    if (t - t_start + 1) % tbptt_k == 0:
                        model.detach_hidden()

                model.detach_hidden()

                # B) Loss Calculation
                preds_super = torch.stack(window_x_preds, dim=1)
                regs_super = torch.stack(window_regs)
                preds_reshaped = preds_super.view(batch_size, J_samples,␣
↪current_window_len, dim_x)

                x_hat_seq = preds_reshaped.mean(dim=1)
                cov_diag_seq = preds_reshaped.var(dim=1) + 1e-9
                target_seq = x_true_batch[:, t_start:t_end, :]

                mse_loss = F.mse_loss(x_hat_seq, target_seq)
                nll_loss = gaussian_nll_safe(target_seq, x_hat_seq,␣
↪cov_diag_seq, max_error_sq=100.0)
                reg_loss = regs_super.mean()

                if train_iter_count < mse_warmup_iters:
                    loss = mse_loss + reg_loss
```

```
                    mode = "MSE_WARMUP"
                else:
                    loss = nll_loss + reg_loss + (lambda_mse * mse_loss)
                    mode = "NLL_OPTIM"

                # C) Update
                loss.backward()
                if clip_grad > 0:
                    torch.nn.utils.clip_grad_norm_(model.parameters(),␣
↪clip_grad)
                optimizer.step()
                optimizer.zero_grad()
                model.detach_hidden()

                train_iter_count += 1

                if train_iter_count % logging_period == 0:
                    with torch.no_grad():
                        diff = x_hat_seq - target_seq
                        mae = diff.abs().mean().item()

                        # Získání hodnot dropoutu (sigmoida z logitů)
                        p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                        p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                    print(f"Iter {train_iter_count} ({mode}): "
                          f"Loss {loss.item():.4f} | "
                          f"MSE {mse_loss.item():.2f} | "
                          f"NLL {nll_loss.item():.2f} | "
                          f"Reg {reg_loss.item():.5f} | " # Regularizační loss
                          f"MAE {mae:.2f}m | "
                          f"p1={p1:.3f}, p2={p2:.3f}")    # Dropout␣
↪pravděpodobnosti
                if train_iter_count % logging_period == 0:
                    with torch.no_grad():
                        diff = x_hat_seq - target_seq
                        mae = diff.abs().mean().item()
                    print(f"Iter {train_iter_count} ({mode}): Loss {loss.item():
↪.4f} | MSE {mse_loss.item():.2f} | NLL {nll_loss.item():.2f} | MAE {mae:.
↪2f}m")

                # --- VALIDATION WITH FULL STATE RESTORE (FIXED) ---
                if train_iter_count % validation_period == 0:
                    # 1. ULOŽIT KOMPLETNÍ STAV TRÉNINKU
                    # Musíme uložit úplně všechno, co se mění v čase t
```

```
                    train_state = {}

                    if model.h_prev is not None:
                        train_state['h_prev'] = model.h_prev.detach().clone()

                    train_state['x_filt_1'] = model.x_filtered_t_minus_1.
↪detach().clone()
                    train_state['x_pred_1'] = model.x_pred_t_minus_1.detach().
↪clone()

                    # !!! ZDE BYLA CHYBA: Musíme uložit i t-2 a y_t-1 !!!
                    if hasattr(model, 'x_filtered_t_minus_2') and model.
↪x_filtered_t_minus_2 is not None:
                        train_state['x_filt_2'] = model.x_filtered_t_minus_2.
↪detach().clone()

                    if hasattr(model, 'y_t_minus_1') and model.y_t_minus_1 is␣
↪not None:
                        train_state['y_1'] = model.y_t_minus_1.detach().clone()

                    if hasattr(model, 'P_t_minus_1') and model.P_t_minus_1 is␣
↪not None:
                        train_state['P'] = model.P_t_minus_1.detach().clone()

                    # 2. Spustit Validaci
                    model.eval()
                    val_mse_list = []
                    all_val_x_true, all_val_x_hat, all_val_P = [], [], []

                    with torch.no_grad():
                        for x_v, y_v in val_loader:
                            x_v, y_v = x_v.to(device), y_v.to(device)
                            b_v, s_v, _ = x_v.shape
                            x_v_sup = x_v.repeat_interleave(J_samples, dim=0)
                            y_v_sup = y_v.repeat_interleave(J_samples, dim=0)

                            # Hard reset pro validaci (smaže interní proměnné)
                            if hasattr(model, 'h_prev'): model.h_prev = None
                            model.reset(batch_size=b_v*J_samples,␣
↪initial_state=x_v_sup[:,0,:])

                            preds_list = []
                            for ti in range(1, s_v):
                                est, _ = model.step(y_v_sup[:, ti, :])
                                preds_list.append(est)
```

```
                            preds_stack = torch.stack(preds_list, dim=1).
↪view(b_v, J_samples, s_v-1, 4)
                            val_mean = preds_stack.mean(dim=1)
                            val_var = preds_stack.var(dim=1) + 1e-9
                            target_v = x_v[:, 1:, :]

                            val_mse_list.append(F.mse_loss(val_mean, target_v).
↪item())

                            # Ukládáme jen srovnatelné tensory (bez t=0)
                            all_val_x_true.append(target_v.cpu()) # Zde byla␣
↪chyba (bylo x_v)
                            all_val_x_hat.append(val_mean.cpu())

                            # P matice musí odpovídat val_mean (tedy bez t=0)
                            val_P_full = torch.zeros(b_v, s_v-1, 4, 4,␣
↪device=device)
                            for i in range(4): val_P_full[:, :, i, i] =␣
↪val_var[:, :, i]
                            # P0 nepřidáváme, protože nemáme odhad pro t=0
                            all_val_P.append(val_P_full.cpu())

                    avg_val_mse = np.mean(val_mse_list)

                    try:
                        cat_true = torch.cat(all_val_x_true, dim=0)
                        cat_hat = torch.cat(all_val_x_hat, dim=0)
                        cat_P = torch.cat(all_val_P, dim=0)
                        # Nyní mají tensory shodnou délku, takže reshape projde
                        avg_anees = calculate_anees_vectorized(cat_true,␣
↪cat_hat, cat_P)
                    except Exception as e:
                        # Vypíšeme chybu, ať víme, co se děje, místo tichého nan
                        print(f"ANEES Error: {e}")
                        avg_anees = float('nan')

                    anees_penalty = abs(avg_anees - 4.0) if not np.
↪isnan(avg_anees) else 100.0

                    # Score = MSE + 10 * ANEES_deviation
                    # Pokud je ANEES 4.0, score = MSE.
                    # Pokud je ANEES 20.0, score = MSE + 160 (výrazné zhoršení).
                    hybrid_score = avg_val_mse + (calibration_parameter *␣
↪anees_penalty)
```

```
                    print(f"\n--- VALIDATION: MSE {avg_val_mse:.2f} | ANEES␣
↪{avg_anees:.2f} | Score {hybrid_score:.2f} ---")

                    # Ukládáme, pokud je hybridní skóre lepší (menší)
                    if hybrid_score < best_val_score:
                        print(f"  >>> New Best Model! (Old Score:␣
↪{best_val_score:.2f} -> New: {hybrid_score:.2f}) <<<")
                        best_val_score = hybrid_score
                        best_model_state = deepcopy(model.state_dict())
                    print("-" * 40)

                    # 3. OBNOVIT KOMPLETNÍ STAV TRÉNINKU
                    model.train()
                    if 'h_prev' in train_state: model.h_prev =␣
↪train_state['h_prev']
                    model.x_filtered_t_minus_1 = train_state['x_filt_1']
                    model.x_pred_t_minus_1 = train_state['x_pred_1']

                    if 'x_filt_2' in train_state: model.x_filtered_t_minus_2 =␣
↪train_state['x_filt_2']
                    if 'y_1' in train_state: model.y_t_minus_1 =␣
↪train_state['y_1']
                    if 'P' in train_state: model.P_t_minus_1 = train_state['P']

    print("Training completed.")
    if best_model_state:
        model.load_state_dict(best_model_state)
    return {"final_model": model}
```

[11]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    safe_var = var + min_var
    error_sq = (preds - target) ** 2
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)
    return nll.mean()

def calculate_anees_internal(x_true, x_hat, P_hat):
    if P_hat.dim() == x_hat.dim(): # Diagonální variance
        error_sq = (x_true - x_hat) ** 2
        nees_per_sample = (error_sq / (P_hat + 1e-9)).sum(dim=-1)
        return nees_per_sample.mean().item()
```

```
    return float('nan')

def train_BayesianKalmanNet_TBPTT_Windowed(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad=1.0,
    J_samples=5,
    tbptt_k=5,
    tbptt_w=20,
    validation_period=50, logging_period=10,
    mse_warmup_iters=0,
    weight_decay_=1e-5,
    lambda_mse=1.0,
    calibration_parameter=10.0
):
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay_)

    best_val_score = float('inf')
    best_model_state = None
    train_iter_count = 0
    done = False

    if not hasattr(model, 'detach_hidden'):
         raise AttributeError("Modelu chybí metoda 'detach_hidden()'.")

    print(f"  START TBPTT Windowed Training (k={tbptt_k}, w={tbptt_w})")
    print(f"    Phase 1: MSE Focused (Soft NLL) (0 - {mse_warmup_iters}␣
 ↪updates)")
    print(f"    Phase 2: NLL Optimization (> {mse_warmup_iters} updates)")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)

            batch_size, seq_len, dim_x = x_true_batch.shape

            x_true_super = x_true_batch.repeat_interleave(J_samples, dim=0)
            y_meas_super = y_meas_batch.repeat_interleave(J_samples, dim=0)
            super_batch_size = x_true_super.shape[0]

            # 1. Hard Reset
            if hasattr(model, 'h_prev'): model.h_prev = None
```

```
            if hasattr(model, 'x_filtered_t_minus_1'): model.
↪x_filtered_t_minus_1 = None
            if hasattr(model, 'x_filtered_t_minus_2'): model.
↪x_filtered_t_minus_2 = None
            model.reset(batch_size=super_batch_size,␣
↪initial_state=x_true_super[:, 0, :])

            # 2. TBPTT Loop
            for t_start in range(1, seq_len, tbptt_w):
                if train_iter_count >= total_train_iter: done = True; break

                t_end = min(t_start + tbptt_w, seq_len)
                if t_end - t_start <= 0: continue

                # Teacher Forcing (20%)
                use_teacher_forcing = (np.random.rand() < 0.2)
                if use_teacher_forcing and t_start > 1:
                    gt_state_prev = x_true_super[:, t_start-1, :]
                    model.x_filtered_t_minus_1 = gt_state_prev.detach().clone()
                    model.x_pred_t_minus_1 = gt_state_prev.detach().clone()
                    model.detach_hidden()
                else:
                    model.detach_hidden()

                optimizer.zero_grad()
                window_x_preds = []
                window_regs = []

                # A) Forward pass
                for t in range(t_start, t_end):
                    y_t = y_meas_super[:, t, :]
                    x_est, reg = model.step(y_t)
                    window_x_preds.append(x_est)
                    window_regs.append(reg)

                    if (t - t_start + 1) % tbptt_k == 0:
                        model.detach_hidden()

                model.detach_hidden()

                # B) Loss Calculation
                preds_super = torch.stack(window_x_preds, dim=1)
                regs_super = torch.stack(window_regs)
                preds_reshaped = preds_super.view(batch_size, J_samples, -1,␣
↪dim_x)

                x_hat_seq = preds_reshaped.mean(dim=1)
```

```
                cov_diag_seq = preds_reshaped.var(dim=1) + 1e-9
                target_seq = x_true_batch[:, t_start:t_end, :]

                mse_loss = F.mse_loss(x_hat_seq, target_seq)
                nll_loss = gaussian_nll_safe(target_seq, x_hat_seq,␣
↪cov_diag_seq, max_error_sq=100.0)
                reg_loss = regs_super.mean()

                # Soft Warmup Strategy
                if train_iter_count < mse_warmup_iters:
                    loss = (1.0 * mse_loss) + nll_loss + reg_loss
                    mode = "MSE_WARMUP"
                else:
                    loss = nll_loss + reg_loss + (lambda_mse * mse_loss)
                    mode = "NLL_OPTIM"

                # C) Update
                loss.backward()
                if clip_grad > 0:
                    torch.nn.utils.clip_grad_norm_(model.parameters(),␣
↪clip_grad)
                optimizer.step()
                optimizer.zero_grad()
                model.detach_hidden()

                train_iter_count += 1

                # --- LOGGING (S PŘIDANOU METRIKOU SIGMA) ---
                if train_iter_count % logging_period == 0:
                    with torch.no_grad():
                        mae = (x_hat_seq - target_seq).abs().mean().item()
                        # Průměrná směrodatná odchylka v metrech␣
↪(sqrt(variance))
                        avg_sigma = cov_diag_seq.mean().sqrt().item()

                        p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                        p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                    print(f"Iter {train_iter_count} ({mode}): "
                          f"Loss {loss.item():.2f} | "
                          f"MSE {mse_loss.item():.2f} | "
                          f"NLL {nll_loss.item():.2f} | "
                          f"Sigma {avg_sigma:.2f}m | " # <--- ZDE VIDÍTE␣
↪NEURČITOST
```

```
                        f"MAE {mae:.2f}m | "
                        f"p1={p1:.2f}, p2={p2:.2f}")

            # --- VALIDATION ---
            if train_iter_count % validation_period == 0:
                train_state = {}
                if model.h_prev is not None: train_state['h_prev'] = model.
↪h_prev.detach().clone()
                train_state['x_filt_1'] = model.x_filtered_t_minus_1.
↪detach().clone()
                train_state['x_pred_1'] = model.x_pred_t_minus_1.detach().
↪clone()
                if hasattr(model, 'x_filtered_t_minus_2') and model.
↪x_filtered_t_minus_2 is not None:
                    train_state['x_filt_2'] = model.x_filtered_t_minus_2.
↪detach().clone()
                if hasattr(model, 'y_t_minus_1') and model.y_t_minus_1 is␣
↪not None:
                    train_state['y_1'] = model.y_t_minus_1.detach().clone()
                if hasattr(model, 'P_t_minus_1') and model.P_t_minus_1 is␣
↪not None:
                    train_state['P'] = model.P_t_minus_1.detach().clone()

                model.eval()
                val_mse_list = []
                all_val_x_true, all_val_x_hat, all_val_P = [], [], []

                # Pro výpočet průměrné sigmy ve validaci
                val_sigma_sum = 0.0
                val_batch_count = 0

                with torch.no_grad():
                    for x_v, y_v in val_loader:
                        x_v, y_v = x_v.to(device), y_v.to(device)
                        b_v, s_v, _ = x_v.shape
                        x_v_sup = x_v.repeat_interleave(J_samples, dim=0)
                        y_v_sup = y_v.repeat_interleave(J_samples, dim=0)

                        if hasattr(model, 'h_prev'): model.h_prev = None
                        model.reset(batch_size=b_v*J_samples,␣
↪initial_state=x_v_sup[:,0,:])

                        preds_list = []
                        for ti in range(1, s_v):
                            est, _ = model.step(y_v_sup[:, ti, :])
                            preds_list.append(est)
```

```
                            preds_stack = torch.stack(preds_list, dim=1).
↪view(b_v, J_samples, s_v-1, 4)
                            val_mean = preds_stack.mean(dim=1)
                            val_var = preds_stack.var(dim=1) + 1e-9
                            target_v = x_v[:, 1:, :]

                            val_mse_list.append(F.mse_loss(val_mean, target_v).
↪item())

                            # Logging sigma stats
                            val_sigma_sum += val_var.mean().sqrt().item()
                            val_batch_count += 1

                            all_val_x_true.append(target_v)
                            all_val_x_hat.append(val_mean)
                            all_val_P.append(val_var)

                    avg_val_mse = np.mean(val_mse_list)
                    avg_val_sigma = val_sigma_sum / max(1, val_batch_count)

                    try:
                        cat_true = torch.cat(all_val_x_true, dim=0)
                        cat_hat = torch.cat(all_val_x_hat, dim=0)
                        cat_P = torch.cat(all_val_P, dim=0)
                        avg_anees = calculate_anees_internal(cat_true, cat_hat,␣
↪cat_P)
                    except Exception as e:
                        print(f"ANEES Error: {e}")
                        avg_anees = 100.0

                    anees_penalty = abs(avg_anees - 4.0)
                    hybrid_score = avg_val_mse + (calibration_parameter *␣
↪anees_penalty)

                    print(f"--- VALIDATION: MSE {avg_val_mse:.2f} | Sigma␣
↪{avg_val_sigma:.2f}m | ANEES {avg_anees:.2f} | Score {hybrid_score:.2f} ---")

                    if hybrid_score < best_val_score:
                        print(f"  >>> New Best Model! (Score: {best_val_score:.
↪2f} -> {hybrid_score:.2f}) <<<")
                        best_val_score = hybrid_score
                        best_model_state = deepcopy(model.state_dict())
                    print("-" * 40)

                    # 3. Restore State
```

```
                    model.train()
                    if 'h_prev' in train_state: model.h_prev =␣
 ↪train_state['h_prev']
                    model.x_filtered_t_minus_1 = train_state['x_filt_1']
                    model.x_pred_t_minus_1 = train_state['x_pred_1']
                    if 'x_filt_2' in train_state: model.x_filtered_t_minus_2 =␣
 ↪train_state['x_filt_2']
                    if 'y_1' in train_state: model.y_t_minus_1 =␣
 ↪train_state['y_1']
                    if 'P' in train_state: model.P_t_minus_1 = train_state['P']

    print("Training completed.")
    if best_model_state:
        model.load_state_dict(best_model_state)
    return {"final_model": model}
```

[12]:
```
import torch
import torch.nn.functional as F
import numpy as np
from copy import deepcopy
import sys

# Ensure utils is importable if needed
# import utils.utils as utils

def gaussian_nll_safe(target, preds, var, min_var=1e-6, max_error_sq=100.0):
    """
    Bezpečná NLL loss funkce.
    """
    safe_var = var + min_var
    error_sq = (preds - target) ** 2
    # Clampujeme velikost chyby v čitateli, aby loss neexplodovala
    error_sq_clamped = torch.clamp(error_sq, max=max_error_sq)
    nll = 0.5 * (torch.log(safe_var) + error_sq_clamped / safe_var)
    return nll.mean()

def train_BayesianKalmanNet_TwoPhase(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad,
    J_samples, validation_period, logging_period,
    mse_warmup_iters=0,
    calibration_parameter=0.0, # <--- NOVÝ PARAMETR: Váha pro ANEES v hybridním␣
 ↪skóre
    weight_decay_=1e-5
):
    # Optimizer
```

```
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,␣
↪weight_decay=weight_decay_)

    # Tracking
    best_hybrid_score = float('inf') # Hlavní metrika pro ukládání
    best_model_state = None
    best_iter_count = 0

    # Výsledky pro return
    best_val_mse = float('inf')
    best_val_anees = float('inf')
    best_val_nll = float('inf')

    train_iter_count = 0
    done = False

    print(f"  START Two-Phase Training")
    print(f"    Phase 1: MSE Warmup (0 - {mse_warmup_iters} iters)")
    print(f"    Phase 2: NLL Optimization ({mse_warmup_iters} -␣
↪{total_train_iter} iters)")
    print(f"    Saving Strategy: Hybrid Score = MSE + ({calibration_parameter}␣
↪* |ANEES - 4.0|)")

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            # Detekce NaN v datech
            if torch.isnan(x_true_batch).any():
                print(f"!!! SKIP BATCH iter {train_iter_count}: NaN found in␣
↪x_true !!!")
                continue

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)
            batch_size, seq_len, _ = x_true_batch.shape

            # --- Training Step ---
            optimizer.zero_grad()

            all_trajectories_for_ensemble = []
            all_regs_for_ensemble = []

            # 1. Ensemble Forward Pass
            for j in range(J_samples):
```

```
                model.reset(batch_size=batch_size, initial_state=x_true_batch[:
↪, 0, :])
                current_trajectory_x_hats = []
                current_trajectory_regs = []

                for t in range(1, seq_len):
                    y_t = y_meas_batch[:, t, :]
                    x_filtered_t, reg_t = model.step(y_t)

                    if torch.isnan(x_filtered_t).any():
                        # Fail-safe pro numerickou nestabilitu
                        print(f"NaN detected in forward pass at iter␣
↪{train_iter_count}")
                        loss = torch.tensor(float('nan'), requires_grad=True) #␣
↪Dummy NaN loss
                        break

                    current_trajectory_x_hats.append(x_filtered_t)
                    current_trajectory_regs.append(reg_t)

                if len(current_trajectory_x_hats) != (seq_len - 1): break #␣
↪Pokud nastal break v inner loop

                all_trajectories_for_ensemble.append(torch.
↪stack(current_trajectory_x_hats, dim=1))
                all_regs_for_ensemble.append(torch.sum(torch.
↪stack(current_trajectory_regs)))

            if len(all_trajectories_for_ensemble) < J_samples:
                 # Pokud nějaký sample selhal, přeskočíme update
                 optimizer.zero_grad()
                 continue

            # 2. Statistiky Ensemble
            ensemble_trajectories = torch.stack(all_trajectories_for_ensemble,␣
↪dim=0)
            x_hat_sequence = ensemble_trajectories.mean(dim=0)

            # Epistemická variance
            cov_diag_sequence = ensemble_trajectories.var(dim=0) + 1e-9

            # Regularizace
            regularization_loss = torch.stack(all_regs_for_ensemble).mean() /␣
↪seq_len

            target_sequence = x_true_batch[:, 1:, :]
```

```
            # --- 3. VÝPOČET LOSS ---
            mse_loss = F.mse_loss(x_hat_sequence, target_sequence)
            nll_loss = gaussian_nll_safe(
                target=target_sequence,
                preds=x_hat_sequence,
                var=cov_diag_sequence
            )

            # Přepínání fází
            mode = ""
            if train_iter_count < mse_warmup_iters:
                loss = mse_loss + regularization_loss
                mode = "Warmup"
            else:
                loss = nll_loss + regularization_loss
                mode = "Optim"

            if torch.isnan(loss):
                print("Collapse detected (NaN loss)"); done = True; break

            loss.backward()

            if clip_grad > 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(), clip_grad)

            optimizer.step()
            train_iter_count += 1

            # --- LOGGING ---
            if train_iter_count % logging_period == 0:
                with torch.no_grad():
                    # Výpočet průměrné směrodatné odchylky v metrech (Sigma)
                    # cov_diag_sequence je variance [m^2], odmocníme pro metry
                    avg_sigma = torch.sqrt(cov_diag_sequence).mean().item()

                    # MAE (Mean Absolute Error) v metrech
                    mae = (x_hat_sequence - target_sequence).abs().mean().item()

                    # Dropout pravděpodobnosti
                    p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                    p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                # Formátovaný výpis dle požadavku
                print(f"Iter {train_iter_count} ({mode}): "
```

```
                    f"Loss {loss.item():.2f} | "
                    f"MSE {mse_loss.item():.2f} | "
                    f"NLL {nll_loss.item():.2f} | "
                    f"Sigma {avg_sigma:.2f}m | "
                    f"MAE {mae:.2f}m | "
                    f"p1={p1:.2f}, p2={p2:.2f}")

        # --- VALIDATION ---
        if train_iter_count > 0 and train_iter_count % validation_period ==␣
↪0:
            print(f"\n--- Validation at iteration {train_iter_count} ---")
            model.eval()
            val_mse_list = []
            val_nll_list = []

            all_val_x_true, all_val_x_hat, all_val_P_hat = [], [], []

            with torch.no_grad():
                for x_true_val, y_meas_val in val_loader:
                    v_bs, v_seq, _ = x_true_val.shape
                    x_true_val = x_true_val.to(device)
                    y_meas_val = y_meas_val.to(device)

                    val_ensemble_trajs = []

                    for j in range(J_samples):
                        model.reset(batch_size=v_bs,␣
↪initial_state=x_true_val[:, 0, :])
                        v_x_hats = []
                        for t in range(1, v_seq):
                            est, _ = model.step(y_meas_val[:, t, :])
                            v_x_hats.append(est)
                        val_ensemble_trajs.append(torch.stack(v_x_hats,␣
↪dim=1))

                    val_ens_stack = torch.stack(val_ensemble_trajs, dim=0)
                    val_mean = val_ens_stack.mean(dim=0)
                    val_var_diag = val_ens_stack.var(dim=0) + 1e-9

                    # Metriky
                    val_mse_list.append(F.mse_loss(val_mean, x_true_val[:,␣
↪1:, :]).item())
                    val_nll_list.append(gaussian_nll_safe(x_true_val[:, 1:,␣
↪:], val_mean, val_var_diag).item())

                    # Data pro ANEES
```

```
                        full_x_hat = torch.cat([x_true_val[:, 0, :].
↪unsqueeze(1), val_mean], dim=1)

                        # Konstrukce plné kovariančí matice P
                        # Použijeme diag_embed pro efektivitu, fallback na␣
↪cyklus
                        try:
                            val_covs_full = torch.diag_embed(val_var_diag)
                        except:
                            val_covs_full = torch.zeros(v_bs, v_seq-1, 4, 4,␣
↪device=device)
                            for b in range(v_bs):
                                for t in range(v_seq-1):
                                    val_covs_full[b, t] = torch.
↪diag(val_var_diag[b, t])

                        P0 = torch.eye(4, device=device).unsqueeze(0).
↪unsqueeze(0).repeat(v_bs, 1, 1, 1) * 1e-6
                        full_P_hat = torch.cat([P0, val_covs_full], dim=1)

                        all_val_x_true.append(x_true_val.cpu())
                        all_val_x_hat.append(full_x_hat.cpu())
                        all_val_P_hat.append(full_P_hat.cpu())

                avg_val_mse = np.mean(val_mse_list)
                avg_val_nll = np.mean(val_nll_list)

                # Výpočet ANEES
                try:
                    # Zde předpokládám, že funkce calculate_anees_vectorized je␣
↪dostupná
                    # buď v 'trainer' nebo importovaná z 'utils'
                    from utils import trainer as tr_utils # Lokální import pro␣
↪jistotu
                    avg_val_anees = tr_utils.calculate_anees_vectorized(
                        torch.cat(all_val_x_true, dim=0),
                        torch.cat(all_val_x_hat, dim=0),
                        torch.cat(all_val_P_hat, dim=0)
                    )
                except:
                    # Fallback pokud import selže, nastavíme na 4.0 (ideál) aby␣
↪nezkazilo skóre
                    # nebo NaN pro info
                    avg_val_anees = 100.0 # Velké číslo aby to bylo vidět
                    # print("Warning: ANEES calculation failed.")
```

```
                # --- HYBRID SCORE CALCULATION ---
                anees_diff = abs(avg_val_anees - 4.0)
                hybrid_score = avg_val_mse + (calibration_parameter *␣
↪anees_diff)

                print(f"  > Val MSE: {avg_val_mse:.4f} | Val ANEES:␣
↪{avg_val_anees:.4f} | Hybrid Score: {hybrid_score:.4f}")

                # Ukládání podle Hybrid Score
                if hybrid_score < best_hybrid_score:
                    print(f"  >>>  New Best Model! (Score: {best_hybrid_score:.
↪4f} -> {hybrid_score:.4f})")
                    best_hybrid_score = hybrid_score
                    best_iter_count = train_iter_count
                    best_model_state = deepcopy(model.state_dict())

                    # Uložíme si metriky tohoto nejlepšího modelu
                    best_val_mse = avg_val_mse
                    best_val_anees = avg_val_anees
                    best_val_nll = avg_val_nll

                print("-" * 60)
                model.train()

    print("\nTraining completed.")
    if best_model_state:
        print(f"Loading best model from iteration {best_iter_count} (Score:␣
↪{best_hybrid_score:.4f})")
        model.load_state_dict(best_model_state)

    # Vracíme slovník výsledků pro uložení
    return {
        "final_model": model,
        "best_iter": best_iter_count,
        "best_val_mse": best_val_mse,
        "best_val_anees": best_val_anees,
        "best_val_nll": best_val_nll
    }
```

[13]:
```
import torch
import torch.nn.functional as F
import numpy as np
from copy import deepcopy
import torch

def calculate_anees_vectorized(x_true: torch.Tensor, x_hat: torch.Tensor, P_hat:
 ↪ torch.Tensor) -> float:
```

```
    """
    Vektorizovaný výpočet ANEES (Average Normalized Estimation Error Squared)
    pro plné kovarianční matice.

    Parametry:
    x_true: Tensor tvaru (Batch, SeqLen, StateDim) - skutečné stavy (Ground␣
↪Truth)
    x_hat: Tensor tvaru (Batch, SeqLen, StateDim) - odhadnuté stavy filtrem
    P_hat: Tensor tvaru (Batch, SeqLen, StateDim, StateDim) - odhadnuté plné␣
↪kovarianční matice

    Návratová hodnota:
    Skalár - průměrná hodnota ANEES (ideálně by se měla blížit StateDim, pro␣
↪TAN to je 4.0)
    """
    # 1. Výpočet chyby odhadu (epsilon)
    error = x_true - x_hat  # Tvar: (B, Seq, Dim)

    # 2. Sloučení Batch a SeqLen dimenzí pro hromadný vektorizovaný výpočet
    B, Seq, Dim = error.shape

    # Přetvarujeme na sloupcové vektory pro lineární algebru: (B*Seq, Dim, 1)
    error_flat = error.reshape(B * Seq, Dim, 1)

    # Přetvarujeme kovariance: (B*Seq, Dim, Dim)
    P_hat_flat = P_hat.reshape(B * Seq, Dim, Dim)

    # 3. Zajištění numerické stability (prevence singulárních matic)
    # Tzv. "Jitter" nebo Tichonovova regularizace - přidáme velmi malé číslo na␣
↪diagonálu
    eye = torch.eye(Dim, device=P_hat.device, dtype=P_hat.dtype).unsqueeze(0).
↪expand(B * Seq, -1, -1)
    P_hat_stable = P_hat_flat + 1e-6 * eye

    # 4. Výpočet P^{-1} * error
    # Matematicky nejstabilnější cesta je vyřešit soustavu rovnic P * y = error␣
↪ =>  y = P^{-1} * error
    try:
        # y má tvar (B*Seq, Dim, 1)
        y = torch.linalg.solve(P_hat_stable, error_flat)
    except RuntimeError as e:
        print(f"  Varování při výpočtu ANEES (Singulární matice zjištěna).␣
↪Fallback na pseudoinverzi: {e}")
        # Bezpečnostní pojistka: pokud by matice i přes regularizaci␣
↪zkolabovala, použijeme SVD pseudoinverzi
        P_inv = torch.linalg.pinv(P_hat_stable)
```

```
        y = torch.bmm(P_inv, error_flat)

    # 5. Výpočet kvadratické formy: error^T * y  (což je error^T * P^{-1} *␣
 ↪error)
    # Transpozice error_flat: (B*Seq, 1, Dim)
    error_flat_T = error_flat.transpose(1, 2)

    # Výsledek maticového násobení (bmm) bude mít tvar (B*Seq, 1, 1)
    nees = torch.bmm(error_flat_T, y)

    # Odstraníme zbytečné dimenze -> (B*Seq,)
    nees = nees.squeeze()

    # 6. ANEES je zprůměrovaná hodnota NEES přes všechny batche a všechny␣
 ↪časové kroky
    anees_value = nees.mean().item()

    return anees_value
import torch
import torch.nn.functional as F
import numpy as np
from copy import deepcopy

def train_BayesianKalmanNet_Hybrid(
    model, train_loader, val_loader, device,
    total_train_iter, learning_rate, clip_grad,
    J_samples, validation_period, logging_period,
    warmup_iterations=0,
    weight_decay_=1e-5,
    lambda_mse=1.0,  # <--- Upozornění: Zde už nikdy nedávej 0.0!
    calibration_parameter=0.0,
    force_seq_len=None,
    add_noise=False
):
    # ------------------------------------------------------------
    # ROZDĚLENÍ PARAMETRŮ PRO OPTIMALIZÁTOR (PER-PARAMETER LR)
    # ------------------------------------------------------------
    p_logits_params = []
    base_params = []

    for name, param in model.named_parameters():
        if 'p_logit' in name:
            p_logits_params.append(param)
        else:
            base_params.append(param)
```

```
    # p_logits dostanou silný fixní learning rate (např. 1e-4) a žádný weight␣
↪decay.
    optimizer = torch.optim.AdamW([
        {'params': base_params, 'lr': learning_rate, 'weight_decay':␣
↪weight_decay_},
        {'params': p_logits_params, 'lr': 1e-3, 'weight_decay': 0.0}
    ])

    # Můžeš zapnout scheduler, pokud chceš plynulejší konvergenci
    # scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    #     optimizer, mode='min', factor=0.5, patience=50, verbose=False
    # )

    best_hybrid_score = float('inf')
    best_iter_count = 0
    best_model_state = None

    best_val_mse = float('inf')
    best_val_anees = float('inf')

    train_iter_count = 0
    done = False

    # --- AGREGÁTORY TRÉNOVACÍCH METRIK ---
    running_loss = 0.0
    running_mse = 0.0
    running_nll = 0.0
    steps_since_val = 0

    print(f"  START Hybrid Training (Faithful SOTA) | Cíl: {total_train_iter}␣
↪iterací")
    if force_seq_len:
        print(f"       Ořez dat: Max délka sekvence = {force_seq_len}")
    if add_noise:
        print(f"       Data Augmentation: ZAPNUTO (Šum v počátečním stavu)")
    print("-" * 100)

    while not done:
        model.train()
        for x_true_batch, y_meas_batch in train_loader:
            if train_iter_count >= total_train_iter: done = True; break

            if force_seq_len is not None and x_true_batch.shape[1] >␣
↪force_seq_len:
                if model.training:
                    # Při tréninku vybereme náhodný úsek (naučí se navigovat␣
↪kdekoliv)
```

```
                    max_start = x_true_batch.shape[1] - force_seq_len
                    start_idx = torch.randint(0, max_start + 1, (1,)).item()
                else:
                    # Při validaci bereme vždy od začátku pro stabilní srovnávání logů
                    start_idx = 0

                x_true_batch = x_true_batch[:, start_idx:start_idx+force_seq_len, :]
                y_meas_batch = y_meas_batch[:, start_idx:start_idx+force_seq_len, :]

            if torch.isnan(x_true_batch).any(): continue

            x_true_batch = x_true_batch.to(device)
            y_meas_batch = y_meas_batch.to(device)
            batch_size, seq_len, state_dim = x_true_batch.shape

            optimizer.zero_grad()

            # ==========================================
            # 1. VEKTORIZOVANÉ MC VZORKOVÁNÍ
            # ==========================================
            mc_batch_size = batch_size * J_samples
            x_true_mc = x_true_batch.repeat(J_samples, 1, 1)
            y_meas_mc = y_meas_batch.repeat(J_samples, 1, 1)

            initial_states = x_true_mc[:, 0, :]

            # DATA AUGMENTATION (Lokální, konzervativní šum)
            if add_noise:
                noise_std = torch.tensor([3.5, 3.5, 0.5, 0.5], device=device).unsqueeze(0).expand_as(initial_states)
                start_noise = torch.randn_like(initial_states) * noise_std
                noisy_initial_states = initial_states + start_noise
            else:
                noisy_initial_states = initial_states

            model.reset(batch_size=mc_batch_size, initial_state=noisy_initial_states)
            current_trajectory_x_hats = []
            current_trajectory_regs = []

            break_flag = False
            for t in range(1, seq_len):
                y_t = y_meas_mc[:, t, :]
                x_filtered_t, reg_t = model.step(y_t)
```

```
                if torch.isnan(x_filtered_t).any():
                    break_flag = True; break

                current_trajectory_x_hats.append(x_filtered_t)
                current_trajectory_regs.append(reg_t)

            if break_flag or len(current_trajectory_x_hats) != (seq_len - 1):
                optimizer.zero_grad(); continue

            # ==========================================
            # 2. ZPRACOVÁNÍ VÝSLEDKŮ
            # ==========================================
            all_x_hats_flat = torch.stack(current_trajectory_x_hats, dim=1)
            all_x_hats = all_x_hats_flat.view(J_samples, batch_size, seq_len -␣
↪1, state_dim)
            x_hat_sequence = all_x_hats.mean(dim=0)

            x_centered = all_x_hats - x_hat_sequence.unsqueeze(0)
            cov_full_sequence = (1.0 / (J_samples - 1)) * torch.
↪einsum('jbtd,jbtk->btdk', x_centered, x_centered)
            # Přidáno malé epsilon 1e-3 pro numerickou stabilitu logaritmu
            cov_diag_sequence = torch.diagonal(cov_full_sequence, dim1=-2,␣
↪dim2=-1) + 1e-3

            regularization_loss = torch.stack(current_trajectory_regs).mean()
            target_sequence = x_true_batch[:, 1:, :]

            # ==========================================
            # 3. VÝPOČET ZTRÁTY (LOSS) - STATE OF THE ART
            # ==========================================

            # A) Klasické MSE - Všechny gradienty pro učení polohy (vah GRU)␣
↪tečou JEN TUDY
            mse_loss = F.mse_loss(x_hat_sequence, target_sequence)

            # B) STOP-GRADIENT (.detach())
            # Odpojíme odhad polohy od NLL, aby si síť nemohla úmyslně␣
↪zhoršovat polohu
            error_sq_detached = (x_hat_sequence.detach() - target_sequence) ** 2

            # C) Výpočet Beta-NLL
            beta = 0.5

            # Vypočítáme NLL člen po členech (element-wise) pomocí odpojené␣
↪chyby
```

```
            nll_elementwise = 0.5 * (torch.log(cov_diag_sequence) +␣
↪error_sq_detached / cov_diag_sequence)

            # D) Zastavení gradientů i u škálovacího faktoru (dle literatury)
            beta_scale = (cov_diag_sequence.detach() ** beta)

            # Výsledná Beta-NLL ztráta (použita čistě pro učení Dropout␣
↪parametrů)
            beta_nll_loss = (nll_elementwise * beta_scale).mean()

            # Celková Loss - perfektní symbióza
            loss = beta_nll_loss + (lambda_mse * mse_loss) +␣
↪regularization_loss * 1.0

            if torch.isnan(loss): done = True; break

            loss.backward()
            if clip_grad > 0: torch.nn.utils.clip_grad_norm_(model.
↪parameters(), clip_grad)
            optimizer.step()

            train_iter_count += 1

            # Akumulace pro čistý výpis (logujeme novou beta_nll_loss)
            running_loss += loss.item()
            running_mse += mse_loss.item()
            running_nll += beta_nll_loss.item()
            steps_since_val += 1

            # ========================================
            # 4. VALIDACE A ČISTÝ VÝPIS LOGŮ
            # ========================================
            if train_iter_count > 0 and train_iter_count % validation_period ==␣
↪0:
                avg_t_loss = running_loss / steps_since_val
                avg_t_mse = running_mse / steps_since_val
                avg_t_nll = running_nll / steps_since_val

                running_loss, running_mse, running_nll, steps_since_val = 0.0,␣
↪0.0, 0.0, 0

                model.eval()
                val_mse_list = []
                val_pos_std_list = []
                all_val_x_true, all_val_x_hat, all_val_P_hat = [], [], []
```

```
                with torch.no_grad():
                    p1 = torch.sigmoid(model.dnn.concrete_dropout1.p_logit).
↪item()
                    p2 = torch.sigmoid(model.dnn.concrete_dropout2.p_logit).
↪item()

                    for x_true_val, y_meas_val in val_loader:
                        if force_seq_len is not None and x_true_val.shape[1] >␣
↪force_seq_len:
                            x_true_val = x_true_val[:, :force_seq_len, :]
                            y_meas_val = y_meas_val[:, :force_seq_len, :]

                        v_bs, v_seq, _ = x_true_val.shape
                        x_true_val = x_true_val.to(device)
                        y_meas_val = y_meas_val.to(device)

                        v_mc_bs = v_bs * J_samples
                        v_x_true_mc = x_true_val.repeat(J_samples, 1, 1)
                        v_y_meas_mc = y_meas_val.repeat(J_samples, 1, 1)

                        model.reset(batch_size=v_mc_bs,␣
↪initial_state=v_x_true_mc[:, 0, :])

                        v_x_hats_list = []
                        for t in range(1, v_seq):
                            est, _ = model.step(v_y_meas_mc[:, t, :])
                            v_x_hats_list.append(est)

                        v_all_x_hats_flat = torch.stack(v_x_hats_list, dim=1)
                        v_all_x_hats = v_all_x_hats_flat.view(J_samples, v_bs,␣
↪v_seq - 1, state_dim)

                        val_mean = v_all_x_hats.mean(dim=0)
                        v_x_centered = v_all_x_hats - val_mean.unsqueeze(0)
                        val_covs_full = (1.0 / (J_samples - 1)) * torch.
↪einsum('jbtd,jbtk->btdk', v_x_centered, v_x_centered)

                        eye_matrix = torch.eye(state_dim, device=device).
↪view(1, 1, state_dim, state_dim)
                        val_covs_full = val_covs_full + eye_matrix * 1e-6

                        val_mse_list.append(F.mse_loss(val_mean, x_true_val[:,␣
↪1:, :]).item())

                        std_px = torch.sqrt(val_covs_full[:, :, 0, 0])
                        std_py = torch.sqrt(val_covs_full[:, :, 1, 1])
```

```
                    batch_pos_std = ((std_px + std_py) / 2.0).mean().item()
                    val_pos_std_list.append(batch_pos_std)

                    full_x_hat = torch.cat([x_true_val[:, 0, :].
↪unsqueeze(1), val_mean], dim=1)
                    P0 = model.system_model.P0.unsqueeze(0).repeat(v_bs, 1,␣
↪1).unsqueeze(1)
                    full_P_hat = torch.cat([P0, val_covs_full], dim=1)

                    all_val_x_true.append(x_true_val.cpu())
                    all_val_x_hat.append(full_x_hat.cpu())
                    all_val_P_hat.append(full_P_hat.cpu())

            avg_val_mse = np.mean(val_mse_list)
            avg_val_pos_std = np.mean(val_pos_std_list)

            try:
                avg_val_anees = calculate_anees_vectorized(
                    torch.cat(all_val_x_true, dim=0),
                    torch.cat(all_val_x_hat, dim=0),
                    torch.cat(all_val_P_hat, dim=0)
                )
            except Exception:
                avg_val_anees = 100.0

            anees_diff = abs(avg_val_anees - state_dim)
            hybrid_score = avg_val_mse + (calibration_parameter *␣
↪anees_diff)

            best_marker = ""
            if hybrid_score < best_hybrid_score:
                best_hybrid_score = hybrid_score
                best_iter_count = train_iter_count
                best_val_mse = avg_val_mse
                best_val_anees = avg_val_anees
                best_model_state = deepcopy(model.state_dict())
                best_marker = "   NEW BEST!"

            mode = "Warmup" if train_iter_count < warmup_iterations else␣
↪"Hybrid"
            print(f"[{train_iter_count:4d}/{total_train_iter}] {mode} | "
                  f"Tr (Loss: {avg_t_loss:.2f}, MSE: {avg_t_mse:.1f},  -NLL:
↪ {avg_t_nll:.2f}) | "
                  f"Val (MSE: {avg_val_mse:.1f}, ANEES: {avg_val_anees:.
↪1f}) | "
```

```
                    f"Diag ( _pos: \033[93m±{avg_val_pos_std:.1f}m\033[0m, p1:
↪ {p1:.2f}, p2: {p2:.2f}){best_marker}")

                model.train()

    print("-" * 100)
    print(f"  Trénink dokončen. Načítám nejlepší model z iterace␣
↪{best_iter_count} "
          f"(Val MSE: {best_val_mse:.4f}, Val ANEES: {best_val_anees:.4f})")

    if best_model_state:
        model.load_state_dict(best_model_state)

    return {"final_model": model, "best_val_mse": best_val_mse,␣
↪"best_val_anees": best_val_anees}
```

[14]:
```
import torch
import copy
import os
import gc # Pro úklid paměti

# --- 1. MAPOVÁNÍ DOSTUPNÝCH DAT (CACHE) ---
seq_len_to_idx = {
    10: 1,
    100: 2,
    300: 3
}

# --- 2. FINÁLNÍ SOTA CURRICULUM ---
curriculum_schedule = [
    {
        'phase_id': 1,
        'target_seq_len': 10,
        'source_data_len': 300,  # Sjednoceno! Bereme rovnou z nejdelších, ale␣
↪ořezáváme
        'iters': 600,
        'lr': 1e-3,
        'lambda_mse': 1.0,       # Startujeme s rovnováhou
        'clip_grad': 1.0,
        'mse_warmup_iters': 600, # V první fázi necháme NLL spát, ať se síť␣
↪naučí fyziku
        'calibration_parameter': 0.0,
        'j_samples': 10
    },
    # {
    #     'phase_id': 2,
    #     'target_seq_len': 25,
```

```
    #     'source_data_len': 300,
    #     'iters': 800,
    #     'lr': 1e-4,
    #     'clip_grad': 0.5,
    #     'lambda_mse': 1.0,        # Už žádná 0.0! Poloha je svatá.
    #     'mse_warmup_iters': 0,    # Odteď běží Stop-Gradient a Beta-NLL naplno
    #     'calibration_parameter': 0.0,
    #     'j_samples': 10
    # },
    {
        'phase_id': 3,
        'target_seq_len': 50,
        'source_data_len': 300,
        'iters': 1000,
        'lr': 5e-5,
        'clip_grad': 0.5,
        'lambda_mse': 2.0,        # Lehce zvýšíme tlak na přesnost polohy
        'mse_warmup_iters': 0,
        'calibration_parameter': 1.0,
        'j_samples': 10
    },
    {
        'phase_id': 4,
        'target_seq_len': 100,
        'source_data_len': 100,
        'iters': 1500,            # Poslední, nejdůležitější fáze (1500 iterací␣
 ↪pro stabilizaci)
        'lr': 1e-5,
        'clip_grad': 0.25,        # Přísnější ořezání gradientů pro jemné ladění␣
 ↪na 100 krocích
        'lambda_mse': 5.0,        # Tvrdý tlak na MSE - síť nesmí uletět
        'mse_warmup_iters': 0,
        'calibration_parameter': 1.0,
        'j_samples': 10
    }
    # FÁZE 5 SMAZÁNA: 100 kroků je pro BPTT strop. Pro generalizaci na 1000␣
 ↪kroků poslouží
    # reset paměti v testovacím skriptu.
]
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"device: {device}")
# --- 3. INICIALIZACE MODELU ---
print("=== INICIALIZACE BKN MODELU ===")
state_knet2 = TAN.StateBayesianKalmanNetTAN(
        system_model=system_model,
        device=device,
        hidden_size_multiplier=8,
```

```
        output_layer_multiplier=4,
        num_gru_layers=1,
        init_max_dropout=0.3, # Skvělý výchozí bod, Beta-NLL si to teď už␣
 ↪zkalibruje samo
        init_min_dropout=0.1
).to(device)

# --- 4. SPUŠTĚNÍ CURRICULA ---
print(f"  START CURRICULUM TRAINING ({len(curriculum_schedule)} fází)")

for phase in curriculum_schedule:
    # --- DŮLEŽITÉ: VYČIŠTĚNÍ PAMĚTI PŘED KAŽDOU FÁZÍ ---
    torch.cuda.empty_cache()
    gc.collect()

    pid = phase['phase_id']
    target_len = phase['target_seq_len']
    source_len = phase['source_data_len']
    current_j_samples = phase.get('j_samples', 5)

    print(f"\n" + "="*60)
    print(f"  FÁZE {pid}: Cílová délka {target_len} (Zdroj dat: {source_len})")
    print(f"    Parametry: LR={phase['lr']}, LambdaMSE={phase['lambda_mse']},␣
 ↪J_samples={current_j_samples}")
    print("="*60)

    try:
        cache_idx = seq_len_to_idx[source_len]
        train_loader = datasets_cache[cache_idx][0]
        val_loader = datasets_cache[cache_idx][1]
        print(f"    -> Načítám loader z datasets_cache[{cache_idx}] (původní␣
 ↪délka dat: {source_len})")
    except KeyError:
        print(f"  CHYBA: Nemám definovaný index v 'seq_len_to_idx' pro délku␣
 ↪{source_len}!")
        break
    except IndexError:
        print(f"  CHYBA: Index {cache_idx} neexistuje v datasets_cache!")
        break

    phase_results = train_BayesianKalmanNet_Hybrid(
        model=state_knet2,
        train_loader=train_loader,
        val_loader=val_loader,
        device=device,

        total_train_iter=phase['iters'],
```

```
        learning_rate=phase['lr'],
        warmup_iterations=phase['mse_warmup_iters'],
        clip_grad=phase['clip_grad'],
        lambda_mse=phase['lambda_mse'],
        calibration_parameter=phase.get('calibration_parameter', 0.0),
        force_seq_len=target_len,

        J_samples=current_j_samples,
        validation_period=10,
        logging_period=1,
        weight_decay_=1e-3,
        add_noise=True  # Konzervativní šum (3.5m) už je nastaven uvnitř funkce!
    )

    if not os.path.exists("checkpoints"): os.makedirs("checkpoints")
    torch.save(state_knet2.state_dict(), f"checkpoints/
 ↪bkn_phase_{pid}_len{target_len}.pth")
    print(f"  Fáze {pid} dokončena. Model uložen.")

print("\n  CELÝ TRÉNINK DOKONČEN.")
```

```
device: cuda
=== INICIALIZACE BKN MODELU ===
initialized with gradient terrain
INFO: Aplikuji 'Start Zero' inicializaci pro Kalman Gain.
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
DEBUG: Výstupní vrstva inicializována konzervativně (interval -0.01 až 0.01).
 START CURRICULUM TRAINING (3 fází)

============================================================
 FÁZE 1: Cílová délka 10 (Zdroj dat: 300)
  Parametry: LR=0.001, LambdaMSE=1.0, J_samples=10
============================================================
  -> Načítám loader z datasets_cache[3] (původní délka dat: 300)
 START Hybrid Training (Faithful SOTA) | Cíl: 600 iterací
     Ořez dat: Max délka sekvence = 10
     Data Augmentation: ZAPNUTO (Šum v počátečním stavu)
--------------------------------------------------------------------------------
```

```
--------------------
[  10/600] Warmup | Tr (Loss: 797.59, MSE: 753.0,  -NLL: 44.62) | Val (MSE:
226.7, ANEES: 20.4) | Diag ( _pos: ±9.5m, p1: 0.23, p2: 0.22)   NEW
BEST!
[  20/600] Warmup | Tr (Loss: 74.97, MSE: 61.0,  -NLL: 13.98) | Val (MSE: 40.9,
ANEES: 14.3) | Diag ( _pos: ±5.9m, p1: 0.23, p2: 0.22)   NEW BEST!
[  30/600] Warmup | Tr (Loss: 26.20, MSE: 17.4,  -NLL: 8.77) | Val (MSE: 21.9,
ANEES: 9.3) | Diag ( _pos: ±5.5m, p1: 0.23, p2: 0.22)   NEW BEST!
[  40/600] Warmup | Tr (Loss: 21.14, MSE: 13.0,  -NLL: 8.09) | Val (MSE: 19.0,
ANEES: 8.4) | Diag ( _pos: ±5.2m, p1: 0.23, p2: 0.22)   NEW BEST!
[  50/600] Warmup | Tr (Loss: 20.80, MSE: 13.4,  -NLL: 7.40) | Val (MSE: 19.8,
ANEES: 8.0) | Diag ( _pos: ±5.3m, p1: 0.23, p2: 0.21)
[  60/600] Warmup | Tr (Loss: 20.88, MSE: 13.6,  -NLL: 7.28) | Val (MSE: 14.6,
ANEES: 8.8) | Diag ( _pos: ±4.7m, p1: 0.22, p2: 0.21)   NEW BEST!
[  70/600] Warmup | Tr (Loss: 19.17, MSE: 12.1,  -NLL: 7.02) | Val (MSE: 13.7,
ANEES: 8.5) | Diag ( _pos: ±4.6m, p1: 0.22, p2: 0.21)   NEW BEST!
[  80/600] Warmup | Tr (Loss: 16.58, MSE: 9.8,  -NLL: 6.78) | Val (MSE: 12.0,
ANEES: 8.3) | Diag ( _pos: ±4.6m, p1: 0.22, p2: 0.21)   NEW BEST!
[  90/600] Warmup | Tr (Loss: 14.97, MSE: 8.4,  -NLL: 6.56) | Val (MSE: 11.1,
ANEES: 8.9) | Diag ( _pos: ±4.4m, p1: 0.22, p2: 0.21)   NEW BEST!
[ 100/600] Warmup | Tr (Loss: 14.52, MSE: 8.1,  -NLL: 6.39) | Val (MSE: 11.9,
ANEES: 9.2) | Diag ( _pos: ±4.2m, p1: 0.22, p2: 0.21)
[ 110/600] Warmup | Tr (Loss: 14.45, MSE: 8.2,  -NLL: 6.27) | Val (MSE: 11.4,
ANEES: 8.9) | Diag ( _pos: ±4.2m, p1: 0.21, p2: 0.21)
[ 120/600] Warmup | Tr (Loss: 16.64, MSE: 10.3,  -NLL: 6.34) | Val (MSE: 11.8,
ANEES: 9.2) | Diag ( _pos: ±4.3m, p1: 0.21, p2: 0.20)
[ 130/600] Warmup | Tr (Loss: 15.07, MSE: 9.0,  -NLL: 6.10) | Val (MSE: 11.4,
ANEES: 9.4) | Diag ( _pos: ±4.2m, p1: 0.21, p2: 0.20)
[ 140/600] Warmup | Tr (Loss: 14.23, MSE: 8.1,  -NLL: 6.09) | Val (MSE: 10.3,
ANEES: 8.9) | Diag ( _pos: ±4.1m, p1: 0.21, p2: 0.20)   NEW BEST!
[ 150/600] Warmup | Tr (Loss: 13.51, MSE: 7.6,  -NLL: 5.94) | Val (MSE: 11.5,
ANEES: 9.5) | Diag ( _pos: ±4.0m, p1: 0.21, p2: 0.20)
[ 160/600] Warmup | Tr (Loss: 15.72, MSE: 9.5,  -NLL: 6.19) | Val (MSE: 10.3,
ANEES: 9.4) | Diag ( _pos: ±4.1m, p1: 0.21, p2: 0.20)   NEW BEST!
[ 170/600] Warmup | Tr (Loss: 13.89, MSE: 8.0,  -NLL: 5.93) | Val (MSE: 10.9,
ANEES: 9.9) | Diag ( _pos: ±4.0m, p1: 0.21, p2: 0.20)
[ 180/600] Warmup | Tr (Loss: 12.89, MSE: 7.1,  -NLL: 5.76) | Val (MSE: 8.9,
ANEES: 9.4) | Diag ( _pos: ±3.9m, p1: 0.20, p2: 0.20)   NEW BEST!
[ 190/600] Warmup | Tr (Loss: 13.08, MSE: 7.3,  -NLL: 5.77) | Val (MSE: 9.8,
ANEES: 9.8) | Diag ( _pos: ±3.9m, p1: 0.20, p2: 0.20)
[ 200/600] Warmup | Tr (Loss: 13.05, MSE: 7.2,  -NLL: 5.89) | Val (MSE: 9.1,
ANEES: 9.2) | Diag ( _pos: ±3.9m, p1: 0.20, p2: 0.20)
[ 210/600] Warmup | Tr (Loss: 18.10, MSE: 12.1,  -NLL: 5.99) | Val (MSE: 9.2,
ANEES: 9.7) | Diag ( _pos: ±3.9m, p1: 0.20, p2: 0.19)
[ 220/600] Warmup | Tr (Loss: 12.50, MSE: 6.7,  -NLL: 5.76) | Val (MSE: 9.1,
ANEES: 9.4) | Diag ( _pos: ±3.8m, p1: 0.20, p2: 0.19)
[ 230/600] Warmup | Tr (Loss: 12.96, MSE: 7.0,  -NLL: 5.91) | Val (MSE: 8.8,
ANEES: 9.4) | Diag ( _pos: ±3.8m, p1: 0.20, p2: 0.19)   NEW BEST!
```

```
[ 240/600] Warmup | Tr (Loss: 11.81, MSE: 6.3, -NLL: 5.56) | Val (MSE: 8.9,
ANEES: 10.6) | Diag ( _pos: ±3.7m, p1: 0.20, p2: 0.19)
[ 250/600] Warmup | Tr (Loss: 11.66, MSE: 6.2, -NLL: 5.45) | Val (MSE: 9.1,
ANEES: 9.8) | Diag ( _pos: ±3.7m, p1: 0.19, p2: 0.19)
[ 260/600] Warmup | Tr (Loss: 11.47, MSE: 6.0, -NLL: 5.43) | Val (MSE: 9.2,
ANEES: 10.2) | Diag ( _pos: ±3.7m, p1: 0.19, p2: 0.19)
[ 270/600] Warmup | Tr (Loss: 12.03, MSE: 6.5, -NLL: 5.57) | Val (MSE: 8.1,
ANEES: 9.9) | Diag ( _pos: ±3.6m, p1: 0.19, p2: 0.19)  NEW BEST!
[ 280/600] Warmup | Tr (Loss: 13.86, MSE: 8.3, -NLL: 5.54) | Val (MSE: 8.1,
ANEES: 9.9) | Diag ( _pos: ±3.7m, p1: 0.19, p2: 0.19)
[ 290/600] Warmup | Tr (Loss: 11.68, MSE: 6.2, -NLL: 5.46) | Val (MSE: 8.4,
ANEES: 9.9) | Diag ( _pos: ±3.7m, p1: 0.19, p2: 0.19)
[ 300/600] Warmup | Tr (Loss: 11.51, MSE: 6.1, -NLL: 5.44) | Val (MSE: 7.5,
ANEES: 10.3) | Diag ( _pos: ±3.5m, p1: 0.19, p2: 0.18)  NEW BEST!
[ 310/600] Warmup | Tr (Loss: 12.83, MSE: 7.4, -NLL: 5.46) | Val (MSE: 8.6,
ANEES: 10.4) | Diag ( _pos: ±3.5m, p1: 0.19, p2: 0.18)
[ 320/600] Warmup | Tr (Loss: 12.03, MSE: 6.6, -NLL: 5.44) | Val (MSE: 7.0,
ANEES: 10.1) | Diag ( _pos: ±3.5m, p1: 0.19, p2: 0.18)  NEW BEST!
[ 330/600] Warmup | Tr (Loss: 12.75, MSE: 7.1, -NLL: 5.69) | Val (MSE: 7.2,
ANEES: 9.9) | Diag ( _pos: ±3.6m, p1: 0.18, p2: 0.18)
[ 340/600] Warmup | Tr (Loss: 12.28, MSE: 6.9, -NLL: 5.36) | Val (MSE: 7.6,
ANEES: 10.6) | Diag ( _pos: ±3.6m, p1: 0.18, p2: 0.18)
[ 350/600] Warmup | Tr (Loss: 12.25, MSE: 6.9, -NLL: 5.34) | Val (MSE: 7.0,
ANEES: 10.2) | Diag ( _pos: ±3.5m, p1: 0.18, p2: 0.18)
[ 360/600] Warmup | Tr (Loss: 10.80, MSE: 5.7, -NLL: 5.06) | Val (MSE: 6.8,
ANEES: 10.0) | Diag ( _pos: ±3.5m, p1: 0.18, p2: 0.18)  NEW BEST!
[ 370/600] Warmup | Tr (Loss: 12.42, MSE: 6.9, -NLL: 5.51) | Val (MSE: 7.3,
ANEES: 9.9) | Diag ( _pos: ±3.6m, p1: 0.18, p2: 0.18)
[ 380/600] Warmup | Tr (Loss: 13.50, MSE: 8.0, -NLL: 5.53) | Val (MSE: 7.2,
ANEES: 9.7) | Diag ( _pos: ±3.5m, p1: 0.18, p2: 0.18)
[ 390/600] Warmup | Tr (Loss: 11.22, MSE: 6.0, -NLL: 5.21) | Val (MSE: 6.8,
ANEES: 9.8) | Diag ( _pos: ±3.4m, p1: 0.18, p2: 0.18)  NEW BEST!
[ 400/600] Warmup | Tr (Loss: 11.80, MSE: 6.5, -NLL: 5.27) | Val (MSE: 6.8,
ANEES: 10.4) | Diag ( _pos: ±3.4m, p1: 0.18, p2: 0.18)
[ 410/600] Warmup | Tr (Loss: 11.17, MSE: 6.0, -NLL: 5.18) | Val (MSE: 6.8,
ANEES: 10.5) | Diag ( _pos: ±3.4m, p1: 0.18, p2: 0.18)
[ 420/600] Warmup | Tr (Loss: 11.90, MSE: 6.6, -NLL: 5.32) | Val (MSE: 7.1,
ANEES: 10.3) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.18)
[ 430/600] Warmup | Tr (Loss: 10.56, MSE: 5.6, -NLL: 4.96) | Val (MSE: 6.9,
ANEES: 10.7) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.18)
[ 440/600] Warmup | Tr (Loss: 11.27, MSE: 6.0, -NLL: 5.25) | Val (MSE: 6.0,
ANEES: 9.8) | Diag ( _pos: ±3.3m, p1: 0.17, p2: 0.18)  NEW BEST!
[ 450/600] Warmup | Tr (Loss: 10.82, MSE: 5.8, -NLL: 5.06) | Val (MSE: 7.3,
ANEES: 10.3) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.18)
[ 460/600] Warmup | Tr (Loss: 11.15, MSE: 6.0, -NLL: 5.16) | Val (MSE: 6.6,
ANEES: 10.3) | Diag ( _pos: ±3.5m, p1: 0.17, p2: 0.18)
[ 470/600] Warmup | Tr (Loss: 11.56, MSE: 6.4, -NLL: 5.11) | Val (MSE: 6.2,
ANEES: 10.3) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.17)
```

```
[ 480/600] Warmup | Tr (Loss: 10.38, MSE: 5.3,  -NLL: 5.04) | Val (MSE: 6.8,
ANEES: 10.5) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.17)
[ 490/600] Warmup | Tr (Loss: 10.44, MSE: 5.4,  -NLL: 4.99) | Val (MSE: 6.2,
ANEES: 35.6) | Diag ( _pos: ±3.3m, p1: 0.17, p2: 0.17)
[ 500/600] Warmup | Tr (Loss: 10.80, MSE: 5.7,  -NLL: 5.07) | Val (MSE: 6.3,
ANEES: 35.6) | Diag ( _pos: ±3.3m, p1: 0.17, p2: 0.17)
[ 510/600] Warmup | Tr (Loss: 10.84, MSE: 5.8,  -NLL: 5.06) | Val (MSE: 6.3,
ANEES: 10.7) | Diag ( _pos: ±3.3m, p1: 0.17, p2: 0.17)
[ 520/600] Warmup | Tr (Loss: 11.08, MSE: 5.9,  -NLL: 5.14) | Val (MSE: 6.0,
ANEES: 10.3) | Diag ( _pos: ±3.4m, p1: 0.17, p2: 0.17)  NEW BEST!
[ 530/600] Warmup | Tr (Loss: 10.12, MSE: 5.2,  -NLL: 4.90) | Val (MSE: 6.0,
ANEES: 10.9) | Diag ( _pos: ±3.1m, p1: 0.16, p2: 0.17)  NEW BEST!
[ 540/600] Warmup | Tr (Loss: 12.10, MSE: 6.9,  -NLL: 5.24) | Val (MSE: 5.9,
ANEES: 10.6) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)  NEW BEST!
[ 550/600] Warmup | Tr (Loss: 10.63, MSE: 5.7,  -NLL: 4.90) | Val (MSE: 6.0,
ANEES: 10.8) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)
[ 560/600] Warmup | Tr (Loss: 10.89, MSE: 5.8,  -NLL: 5.11) | Val (MSE: 6.1,
ANEES: 10.3) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)
[ 570/600] Warmup | Tr (Loss: 10.24, MSE: 5.3,  -NLL: 4.98) | Val (MSE: 6.4,
ANEES: 11.0) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)
[ 580/600] Warmup | Tr (Loss: 9.73, MSE: 5.0,  -NLL: 4.75) | Val (MSE: 6.3,
ANEES: 10.2) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)
[ 590/600] Warmup | Tr (Loss: 10.22, MSE: 5.3,  -NLL: 4.87) | Val (MSE: 5.5,
ANEES: 10.2) | Diag ( _pos: ±3.1m, p1: 0.16, p2: 0.17)  NEW BEST!
[ 600/600] Hybrid | Tr (Loss: 10.42, MSE: 5.5,  -NLL: 4.96) | Val (MSE: 6.2,
ANEES: 10.0) | Diag ( _pos: ±3.2m, p1: 0.16, p2: 0.17)
------------------------------------------------------------------------------
--------------------
 Trénink dokončen. Načítám nejlepší model z iterace 590 (Val MSE: 5.4962, Val
ANEES: 10.2347)
 Fáze 1 dokončena. Model uložen.

============================================================
 FÁZE 3: Cílová délka 50 (Zdroj dat: 300)
  Parametry: LR=5e-05, LambdaMSE=2.0, J_samples=10
============================================================
  -> Načítám loader z datasets_cache[3] (původní délka dat: 300)
 START Hybrid Training (Faithful SOTA) | Cíl: 1000 iterací
      Ořez dat: Max délka sekvence = 50
      Data Augmentation: ZAPNUTO (Šum v počátečním stavu)
------------------------------------------------------------------------------
--------------------
[  10/1000] Hybrid | Tr (Loss: 113.47, MSE: 50.0,  -NLL: 13.47) | Val (MSE:
30.0, ANEES: 10.7) | Diag ( _pos: ±7.8m, p1: 0.16, p2: 0.17)  NEW
BEST!
[  20/1000] Hybrid | Tr (Loss: 95.13, MSE: 41.0,  -NLL: 13.13) | Val (MSE: 38.8,
ANEES: 10.5) | Diag ( _pos: ±8.1m, p1: 0.16, p2: 0.17)
[  30/1000] Hybrid | Tr (Loss: 70.06, MSE: 29.3,  -NLL: 11.36) | Val (MSE: 39.6,
```

```
ANEES: 10.8) | Diag ( _pos: ±7.9m, p1: 0.16, p2: 0.17)
[  40/1000] Hybrid | Tr (Loss: 63.45, MSE: 26.3,  -NLL: 10.86) | Val (MSE: 47.2,
ANEES: 10.7) | Diag ( _pos: ±8.4m, p1: 0.16, p2: 0.17)
[  50/1000] Hybrid | Tr (Loss: 86.48, MSE: 37.1,  -NLL: 12.24) | Val (MSE: 57.9,
ANEES: 10.9) | Diag ( _pos: ±7.9m, p1: 0.15, p2: 0.16)
[  60/1000] Hybrid | Tr (Loss: 63.40, MSE: 26.0,  -NLL: 11.31) | Val (MSE: 24.5,
ANEES: 10.8) | Diag ( _pos: ±7.3m, p1: 0.15, p2: 0.16)  NEW BEST!
[  70/1000] Hybrid | Tr (Loss: 67.16, MSE: 28.1,  -NLL: 10.87) | Val (MSE: 28.0,
ANEES: 11.3) | Diag ( _pos: ±7.5m, p1: 0.15, p2: 0.16)
[  80/1000] Hybrid | Tr (Loss: 73.39, MSE: 30.9,  -NLL: 11.55) | Val (MSE: 33.9,
ANEES: 11.0) | Diag ( _pos: ±7.7m, p1: 0.15, p2: 0.16)
[  90/1000] Hybrid | Tr (Loss: 79.60, MSE: 33.9,  -NLL: 11.89) | Val (MSE: 36.1,
ANEES: 11.1) | Diag ( _pos: ±7.6m, p1: 0.15, p2: 0.16)
[ 100/1000] Hybrid | Tr (Loss: 89.99, MSE: 39.0,  -NLL: 11.94) | Val (MSE: 33.1,
ANEES: 11.1) | Diag ( _pos: ±7.4m, p1: 0.15, p2: 0.16)
[ 110/1000] Hybrid | Tr (Loss: 59.96, MSE: 24.8,  -NLL: 10.42) | Val (MSE: 27.8,
ANEES: 11.1) | Diag ( _pos: ±7.4m, p1: 0.15, p2: 0.16)
[ 120/1000] Hybrid | Tr (Loss: 97.28, MSE: 42.4,  -NLL: 12.56) | Val (MSE: 23.4,
ANEES: 11.4) | Diag ( _pos: ±7.1m, p1: 0.15, p2: 0.16)  NEW BEST!
[ 130/1000] Hybrid | Tr (Loss: 76.26, MSE: 32.5,  -NLL: 11.20) | Val (MSE: 37.6,
ANEES: 11.4) | Diag ( _pos: ±7.6m, p1: 0.15, p2: 0.16)
[ 140/1000] Hybrid | Tr (Loss: 56.11, MSE: 23.0,  -NLL: 10.16) | Val (MSE: 23.1,
ANEES: 11.4) | Diag ( _pos: ±7.0m, p1: 0.15, p2: 0.16)  NEW BEST!
[ 150/1000] Hybrid | Tr (Loss: 56.14, MSE: 22.9,  -NLL: 10.42) | Val (MSE: 22.4,
ANEES: 11.3) | Diag ( _pos: ±7.0m, p1: 0.14, p2: 0.16)  NEW BEST!
[ 160/1000] Hybrid | Tr (Loss: 74.11, MSE: 31.4,  -NLL: 11.27) | Val (MSE: 23.0,
ANEES: 11.5) | Diag ( _pos: ±7.0m, p1: 0.14, p2: 0.15)
[ 170/1000] Hybrid | Tr (Loss: 75.45, MSE: 32.0,  -NLL: 11.49) | Val (MSE: 34.1,
ANEES: 11.7) | Diag ( _pos: ±7.2m, p1: 0.14, p2: 0.15)
[ 180/1000] Hybrid | Tr (Loss: 63.87, MSE: 26.7,  -NLL: 10.56) | Val (MSE: 32.1,
ANEES: 11.8) | Diag ( _pos: ±7.6m, p1: 0.14, p2: 0.15)
[ 190/1000] Hybrid | Tr (Loss: 103.64, MSE: 45.2,  -NLL: 13.16) | Val (MSE:
40.4, ANEES: 11.7) | Diag ( _pos: ±7.7m, p1: 0.14, p2: 0.15)
[ 200/1000] Hybrid | Tr (Loss: 73.33, MSE: 31.0,  -NLL: 11.41) | Val (MSE: 22.6,
ANEES: 11.6) | Diag ( _pos: ±6.9m, p1: 0.14, p2: 0.15)
[ 210/1000] Hybrid | Tr (Loss: 55.20, MSE: 22.6,  -NLL: 10.01) | Val (MSE: 49.5,
ANEES: 11.9) | Diag ( _pos: ±8.4m, p1: 0.14, p2: 0.15)
[ 220/1000] Hybrid | Tr (Loss: 59.12, MSE: 24.3,  -NLL: 10.61) | Val (MSE: 22.8,
ANEES: 11.9) | Diag ( _pos: ±6.8m, p1: 0.14, p2: 0.15)
[ 230/1000] Hybrid | Tr (Loss: 55.97, MSE: 23.0,  -NLL: 9.87) | Val (MSE: 34.9,
ANEES: 12.0) | Diag ( _pos: ±7.4m, p1: 0.14, p2: 0.15)
[ 240/1000] Hybrid | Tr (Loss: 52.90, MSE: 21.5,  -NLL: 9.86) | Val (MSE: 35.8,
ANEES: 12.2) | Diag ( _pos: ±7.6m, p1: 0.14, p2: 0.15)
[ 250/1000] Hybrid | Tr (Loss: 57.71, MSE: 23.8,  -NLL: 10.10) | Val (MSE: 22.7,
ANEES: 12.3) | Diag ( _pos: ±6.7m, p1: 0.14, p2: 0.15)
[ 260/1000] Hybrid | Tr (Loss: 67.26, MSE: 28.1,  -NLL: 10.97) | Val (MSE: 41.8,
ANEES: 12.4) | Diag ( _pos: ±7.6m, p1: 0.14, p2: 0.15)
[ 270/1000] Hybrid | Tr (Loss: 54.69, MSE: 22.3,  -NLL: 10.00) | Val (MSE: 22.1,
```

```
ANEES: 12.0) | Diag ( _pos: ±6.8m, p1: 0.14, p2: 0.15)
[ 280/1000] Hybrid | Tr (Loss: 78.14, MSE: 33.4, -NLL: 11.24) | Val (MSE: 21.5,
ANEES: 12.3) | Diag ( _pos: ±6.7m, p1: 0.14, p2: 0.15)
[ 290/1000] Hybrid | Tr (Loss: 51.01, MSE: 20.6, -NLL: 9.74) | Val (MSE: 37.8,
ANEES: 12.1) | Diag ( _pos: ±7.4m, p1: 0.14, p2: 0.15)
[ 300/1000] Hybrid | Tr (Loss: 52.83, MSE: 21.4, -NLL: 9.95) | Val (MSE: 21.6,
ANEES: 11.8) | Diag ( _pos: ±6.8m, p1: 0.13, p2: 0.15)  NEW BEST!
[ 310/1000] Hybrid | Tr (Loss: 71.64, MSE: 30.4, -NLL: 10.91) | Val (MSE: 29.3,
ANEES: 12.1) | Diag ( _pos: ±7.2m, p1: 0.13, p2: 0.15)
[ 320/1000] Hybrid | Tr (Loss: 120.93, MSE: 54.8, -NLL: 11.42) | Val (MSE:
23.1, ANEES: 12.6) | Diag ( _pos: ±6.8m, p1: 0.13, p2: 0.15)
[ 330/1000] Hybrid | Tr (Loss: 51.90, MSE: 21.2, -NLL: 9.56) | Val (MSE: 26.9,
ANEES: 12.5) | Diag ( _pos: ±7.0m, p1: 0.13, p2: 0.15)
[ 340/1000] Hybrid | Tr (Loss: 72.77, MSE: 31.0, -NLL: 10.86) | Val (MSE: 25.3,
ANEES: 12.4) | Diag ( _pos: ±7.1m, p1: 0.13, p2: 0.15)
[ 350/1000] Hybrid | Tr (Loss: 53.93, MSE: 22.0, -NLL: 9.93) | Val (MSE: 35.6,
ANEES: 12.4) | Diag ( _pos: ±7.3m, p1: 0.13, p2: 0.15)
[ 360/1000] Hybrid | Tr (Loss: 54.90, MSE: 22.5, -NLL: 9.83) | Val (MSE: 21.6,
ANEES: 12.2) | Diag ( _pos: ±6.7m, p1: 0.13, p2: 0.14)
[ 370/1000] Hybrid | Tr (Loss: 74.09, MSE: 31.5, -NLL: 11.17) | Val (MSE: 22.2,
ANEES: 12.3) | Diag ( _pos: ±6.6m, p1: 0.13, p2: 0.14)
[ 380/1000] Hybrid | Tr (Loss: 53.93, MSE: 22.1, -NLL: 9.65) | Val (MSE: 22.0,
ANEES: 12.5) | Diag ( _pos: ±6.6m, p1: 0.13, p2: 0.14)
[ 390/1000] Hybrid | Tr (Loss: 105.20, MSE: 46.8, -NLL: 11.63) | Val (MSE:
22.7, ANEES: 12.6) | Diag ( _pos: ±6.7m, p1: 0.13, p2: 0.14)
[ 400/1000] Hybrid | Tr (Loss: 53.20, MSE: 21.8, -NLL: 9.67) | Val (MSE: 22.3,
ANEES: 12.6) | Diag ( _pos: ±6.6m, p1: 0.13, p2: 0.14)
[ 410/1000] Hybrid | Tr (Loss: 122.83, MSE: 56.1, -NLL: 10.63) | Val (MSE:
25.8, ANEES: 12.3) | Diag ( _pos: ±6.8m, p1: 0.13, p2: 0.14)
[ 420/1000] Hybrid | Tr (Loss: 53.82, MSE: 22.1, -NLL: 9.70) | Val (MSE: 36.3,
ANEES: 12.5) | Diag ( _pos: ±7.4m, p1: 0.13, p2: 0.14)
[ 430/1000] Hybrid | Tr (Loss: 64.85, MSE: 27.3, -NLL: 10.33) | Val (MSE: 22.1,
ANEES: 12.7) | Diag ( _pos: ±6.5m, p1: 0.13, p2: 0.14)
[ 440/1000] Hybrid | Tr (Loss: 83.48, MSE: 35.8, -NLL: 11.92) | Val (MSE: 22.5,
ANEES: 12.7) | Diag ( _pos: ±6.5m, p1: 0.13, p2: 0.14)
[ 450/1000] Hybrid | Tr (Loss: 59.16, MSE: 24.5, -NLL: 10.25) | Val (MSE: 46.7,
ANEES: 12.8) | Diag ( _pos: ±7.7m, p1: 0.13, p2: 0.14)
[ 460/1000] Hybrid | Tr (Loss: 107.95, MSE: 48.5, -NLL: 10.86) | Val (MSE:
21.4, ANEES: 13.0) | Diag ( _pos: ±6.5m, p1: 0.13, p2: 0.14)
[ 470/1000] Hybrid | Tr (Loss: 55.67, MSE: 23.0, -NLL: 9.61) | Val (MSE: 34.8,
ANEES: 12.7) | Diag ( _pos: ±6.7m, p1: 0.12, p2: 0.14)
[ 480/1000] Hybrid | Tr (Loss: 51.33, MSE: 20.9, -NLL: 9.49) | Val (MSE: 34.1,
ANEES: 12.9) | Diag ( _pos: ±6.8m, p1: 0.12, p2: 0.14)
[ 490/1000] Hybrid | Tr (Loss: 50.85, MSE: 20.7, -NLL: 9.35) | Val (MSE: 22.7,
ANEES: 13.0) | Diag ( _pos: ±6.6m, p1: 0.12, p2: 0.14)
[ 500/1000] Hybrid | Tr (Loss: 61.41, MSE: 25.8, -NLL: 9.83) | Val (MSE: 34.7,
ANEES: 13.4) | Diag ( _pos: ±7.1m, p1: 0.12, p2: 0.14)
[ 510/1000] Hybrid | Tr (Loss: 56.69, MSE: 23.6, -NLL: 9.59) | Val (MSE: 21.9,
```

```
ANEES: 12.9) | Diag ( _pos: ±6.5m, p1: 0.12, p2: 0.14)
[ 520/1000] Hybrid | Tr (Loss: 66.09, MSE: 27.8, -NLL: 10.50) | Val (MSE: 57.7,
ANEES: 12.5) | Diag ( _pos: ±7.2m, p1: 0.12, p2: 0.14)
[ 530/1000] Hybrid | Tr (Loss: 70.71, MSE: 30.0, -NLL: 10.67) | Val (MSE: 35.5,
ANEES: 17.9) | Diag ( _pos: ±7.2m, p1: 0.12, p2: 0.14)
[ 540/1000] Hybrid | Tr (Loss: 58.75, MSE: 24.4, -NLL: 10.02) | Val (MSE: 20.8,
ANEES: 13.2) | Diag ( _pos: ±6.4m, p1: 0.12, p2: 0.14)
[ 550/1000] Hybrid | Tr (Loss: 55.52, MSE: 22.8, -NLL: 9.89) | Val (MSE: 24.8,
ANEES: 12.9) | Diag ( _pos: ±6.8m, p1: 0.12, p2: 0.14)
[ 560/1000] Hybrid | Tr (Loss: 56.07, MSE: 23.2, -NLL: 9.64) | Val (MSE: 21.9,
ANEES: 13.0) | Diag ( _pos: ±6.4m, p1: 0.12, p2: 0.14)
[ 570/1000] Hybrid | Tr (Loss: 54.50, MSE: 22.5, -NLL: 9.52) | Val (MSE: 21.1,
ANEES: 12.9) | Diag ( _pos: ±6.3m, p1: 0.12, p2: 0.14)
[ 580/1000] Hybrid | Tr (Loss: 82.42, MSE: 35.5, -NLL: 11.48) | Val (MSE: 32.9,
ANEES: 13.1) | Diag ( _pos: ±7.0m, p1: 0.12, p2: 0.14)
[ 590/1000] Hybrid | Tr (Loss: 63.16, MSE: 26.4, -NLL: 10.38) | Val (MSE: 28.9,
ANEES: 12.8) | Diag ( _pos: ±6.9m, p1: 0.12, p2: 0.14)
[ 600/1000] Hybrid | Tr (Loss: 79.58, MSE: 34.2, -NLL: 11.15) | Val (MSE: 30.9,
ANEES: 12.9) | Diag ( _pos: ±6.9m, p1: 0.12, p2: 0.14)
[ 610/1000] Hybrid | Tr (Loss: 58.13, MSE: 24.1, -NLL: 9.83) | Val (MSE: 32.9,
ANEES: 13.1) | Diag ( _pos: ±6.9m, p1: 0.12, p2: 0.14)
[ 620/1000] Hybrid | Tr (Loss: 50.26, MSE: 20.4, -NLL: 9.38) | Val (MSE: 23.1,
ANEES: 13.0) | Diag ( _pos: ±6.4m, p1: 0.12, p2: 0.14)
[ 630/1000] Hybrid | Tr (Loss: 55.71, MSE: 23.1, -NLL: 9.44) | Val (MSE: 30.1,
ANEES: 12.9) | Diag ( _pos: ±6.7m, p1: 0.12, p2: 0.14)
[ 640/1000] Hybrid | Tr (Loss: 75.98, MSE: 32.4, -NLL: 11.10) | Val (MSE: 20.8,
ANEES: 13.0) | Diag ( _pos: ±6.3m, p1: 0.12, p2: 0.14)
[ 650/1000] Hybrid | Tr (Loss: 57.29, MSE: 23.8, -NLL: 9.66) | Val (MSE: 31.4,
ANEES: 13.3) | Diag ( _pos: ±6.9m, p1: 0.12, p2: 0.14)
[ 660/1000] Hybrid | Tr (Loss: 58.09, MSE: 24.1, -NLL: 9.89) | Val (MSE: 20.8,
ANEES: 13.1) | Diag ( _pos: ±6.3m, p1: 0.12, p2: 0.14)
[ 670/1000] Hybrid | Tr (Loss: 61.48, MSE: 25.7, -NLL: 10.04) | Val (MSE: 24.6,
ANEES: 13.3) | Diag ( _pos: ±6.6m, p1: 0.12, p2: 0.14)
[ 680/1000] Hybrid | Tr (Loss: 65.16, MSE: 27.3, -NLL: 10.53) | Val (MSE: 21.8,
ANEES: 13.1) | Diag ( _pos: ±6.4m, p1: 0.12, p2: 0.14)
[ 690/1000] Hybrid | Tr (Loss: 88.76, MSE: 39.0, -NLL: 10.84) | Val (MSE: 20.8,
ANEES: 13.5) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 700/1000] Hybrid | Tr (Loss: 56.13, MSE: 23.2, -NLL: 9.64) | Val (MSE: 25.3,
ANEES: 13.5) | Diag ( _pos: ±6.6m, p1: 0.11, p2: 0.14)
[ 710/1000] Hybrid | Tr (Loss: 66.76, MSE: 28.3, -NLL: 10.06) | Val (MSE: 24.9,
ANEES: 13.0) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 720/1000] Hybrid | Tr (Loss: 64.69, MSE: 27.3, -NLL: 10.09) | Val (MSE: 22.0,
ANEES: 13.5) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 730/1000] Hybrid | Tr (Loss: 62.61, MSE: 26.2, -NLL: 10.29) | Val (MSE: 24.9,
ANEES: 13.2) | Diag ( _pos: ±6.6m, p1: 0.11, p2: 0.14)
[ 740/1000] Hybrid | Tr (Loss: 57.22, MSE: 23.8, -NLL: 9.57) | Val (MSE: 24.8,
ANEES: 13.3) | Diag ( _pos: ±6.5m, p1: 0.11, p2: 0.14)
[ 750/1000] Hybrid | Tr (Loss: 56.72, MSE: 23.5, -NLL: 9.69) | Val (MSE: 26.3,
```

```
ANEES: 13.0) | Diag ( _pos: ±6.6m, p1: 0.11, p2: 0.14)
[ 760/1000] Hybrid | Tr (Loss: 63.09, MSE: 26.5, -NLL: 10.12) | Val (MSE: 30.0,
ANEES: 13.1) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 770/1000] Hybrid | Tr (Loss: 87.06, MSE: 37.8, -NLL: 11.48) | Val (MSE: 21.0,
ANEES: 12.8) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 780/1000] Hybrid | Tr (Loss: 61.41, MSE: 25.7, -NLL: 9.93) | Val (MSE: 24.3,
ANEES: 12.9) | Diag ( _pos: ±6.6m, p1: 0.11, p2: 0.14)
[ 790/1000] Hybrid | Tr (Loss: 177.79, MSE: 83.4, -NLL: 10.97) | Val (MSE:
30.7, ANEES: 13.4) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 800/1000] Hybrid | Tr (Loss: 68.26, MSE: 29.0, -NLL: 10.28) | Val (MSE: 36.6,
ANEES: 13.8) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 810/1000] Hybrid | Tr (Loss: 69.18, MSE: 29.4, -NLL: 10.45) | Val (MSE: 21.9,
ANEES: 13.3) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 820/1000] Hybrid | Tr (Loss: 69.82, MSE: 29.6, -NLL: 10.56) | Val (MSE: 21.1,
ANEES: 13.1) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 830/1000] Hybrid | Tr (Loss: 58.27, MSE: 24.4, -NLL: 9.57) | Val (MSE: 21.0,
ANEES: 23.4) | Diag ( _pos: ±6.2m, p1: 0.11, p2: 0.14)
[ 840/1000] Hybrid | Tr (Loss: 80.87, MSE: 34.8, -NLL: 11.33) | Val (MSE: 20.9,
ANEES: 13.4) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 850/1000] Hybrid | Tr (Loss: 51.89, MSE: 21.3, -NLL: 9.31) | Val (MSE: 33.5,
ANEES: 13.0) | Diag ( _pos: ±6.6m, p1: 0.11, p2: 0.14)
[ 860/1000] Hybrid | Tr (Loss: 53.67, MSE: 22.2, -NLL: 9.29) | Val (MSE: 21.1,
ANEES: 13.0) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 870/1000] Hybrid | Tr (Loss: 91.25, MSE: 40.4, -NLL: 10.42) | Val (MSE: 20.8,
ANEES: 13.2) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 880/1000] Hybrid | Tr (Loss: 51.56, MSE: 21.1, -NLL: 9.30) | Val (MSE: 28.5,
ANEES: 12.9) | Diag ( _pos: ±6.8m, p1: 0.11, p2: 0.14)
[ 890/1000] Hybrid | Tr (Loss: 50.78, MSE: 20.8, -NLL: 9.11) | Val (MSE: 20.9,
ANEES: 13.1) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 900/1000] Hybrid | Tr (Loss: 53.36, MSE: 22.0, -NLL: 9.39) | Val (MSE: 28.2,
ANEES: 12.7) | Diag ( _pos: ±6.8m, p1: 0.11, p2: 0.14)
[ 910/1000] Hybrid | Tr (Loss: 51.69, MSE: 21.2, -NLL: 9.22) | Val (MSE: 20.8,
ANEES: 13.0) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 920/1000] Hybrid | Tr (Loss: 52.12, MSE: 21.5, -NLL: 9.21) | Val (MSE: 25.3,
ANEES: 18.4) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 930/1000] Hybrid | Tr (Loss: 52.55, MSE: 21.6, -NLL: 9.41) | Val (MSE: 21.4,
ANEES: 13.0) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[ 940/1000] Hybrid | Tr (Loss: 49.74, MSE: 20.3, -NLL: 9.18) | Val (MSE: 29.8,
ANEES: 13.0) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 950/1000] Hybrid | Tr (Loss: 73.26, MSE: 31.2, -NLL: 10.96) | Val (MSE: 30.0,
ANEES: 17.8) | Diag ( _pos: ±6.7m, p1: 0.11, p2: 0.14)
[ 960/1000] Hybrid | Tr (Loss: 50.88, MSE: 20.8, -NLL: 9.23) | Val (MSE: 20.6,
ANEES: 12.9) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 970/1000] Hybrid | Tr (Loss: 81.00, MSE: 34.8, -NLL: 11.43) | Val (MSE: 23.8,
ANEES: 12.9) | Diag ( _pos: ±6.5m, p1: 0.11, p2: 0.14)
[ 980/1000] Hybrid | Tr (Loss: 47.78, MSE: 19.4, -NLL: 9.01) | Val (MSE: 24.6,
ANEES: 13.0) | Diag ( _pos: ±6.4m, p1: 0.11, p2: 0.14)
[ 990/1000] Hybrid | Tr (Loss: 54.33, MSE: 22.4, -NLL: 9.47) | Val (MSE: 21.2,
```

```
ANEES: 13.3) | Diag ( _pos: ±6.3m, p1: 0.11, p2: 0.14)
[1000/1000] Hybrid | Tr (Loss: 55.56, MSE: 23.0, -NLL: 9.51) | Val (MSE: 19.6,
ANEES: 12.9) | Diag ( _pos: ±6.2m, p1: 0.11, p2: 0.14)  NEW BEST!
--------------------------------------------------------------------------------
--------------------
 Trénink dokončen. Načítám nejlepší model z iterace 1000 (Val MSE: 19.5979, Val
ANEES: 12.9018)
 Fáze 3 dokončena. Model uložen.

============================================================
 FÁZE 4: Cílová délka 100 (Zdroj dat: 100)
  Parametry: LR=1e-05, LambdaMSE=5.0, J_samples=10
============================================================
  -> Načítám loader z datasets_cache[2] (původní délka dat: 100)
 START Hybrid Training (Faithful SOTA) | Cíl: 1500 iterací
      Ořez dat: Max délka sekvence = 100
      Data Augmentation: ZAPNUTO (Šum v počátečním stavu)
--------------------------------------------------------------------------------
--------------------
[  10/1500] Hybrid | Tr (Loss: 1594.78, MSE: 315.4, -NLL: 17.89) | Val (MSE:
43.0, ANEES: 13.2) | Diag ( _pos: ±9.0m, p1: 0.10, p2: 0.14)  NEW
BEST!
[  20/1500] Hybrid | Tr (Loss: 2941.07, MSE: 582.7, -NLL: 27.74) | Val (MSE:
43.0, ANEES: 13.1) | Diag ( _pos: ±8.9m, p1: 0.10, p2: 0.14)  NEW
BEST!
[  30/1500] Hybrid | Tr (Loss: 408.91, MSE: 78.5, -NLL: 16.51) | Val (MSE:
43.8, ANEES: 13.4) | Diag ( _pos: ±8.9m, p1: 0.10, p2: 0.14)
[  40/1500] Hybrid | Tr (Loss: 427.13, MSE: 82.1, -NLL: 16.81) | Val (MSE:
43.6, ANEES: 13.4) | Diag ( _pos: ±9.1m, p1: 0.10, p2: 0.14)
[  50/1500] Hybrid | Tr (Loss: 508.54, MSE: 97.9, -NLL: 19.15) | Val (MSE:
44.4, ANEES: 13.8) | Diag ( _pos: ±8.8m, p1: 0.10, p2: 0.14)
[  60/1500] Hybrid | Tr (Loss: 282.28, MSE: 53.5, -NLL: 15.01) | Val (MSE:
68.3, ANEES: 13.9) | Diag ( _pos: ±9.5m, p1: 0.10, p2: 0.14)
[  70/1500] Hybrid | Tr (Loss: 7748.95, MSE: 1541.7, -NLL: 40.66) | Val (MSE:
118.1, ANEES: 13.8) | Diag ( _pos: ±10.6m, p1: 0.10, p2: 0.14)
[  80/1500] Hybrid | Tr (Loss: 3825.40, MSE: 760.3, -NLL: 23.83) | Val (MSE:
184.7, ANEES: 13.9) | Diag ( _pos: ±10.0m, p1: 0.10, p2: 0.14)
[  90/1500] Hybrid | Tr (Loss: 4936.31, MSE: 983.0, -NLL: 21.40) | Val (MSE:
47.5, ANEES: 14.1) | Diag ( _pos: ±8.8m, p1: 0.10, p2: 0.14)
[ 100/1500] Hybrid | Tr (Loss: 409.38, MSE: 78.5, -NLL: 16.98) | Val (MSE:
84.6, ANEES: 13.9) | Diag ( _pos: ±9.9m, p1: 0.10, p2: 0.14)
[ 110/1500] Hybrid | Tr (Loss: 324.02, MSE: 61.6, -NLL: 16.12) | Val (MSE:
58.6, ANEES: 16.7) | Diag ( _pos: ±9.1m, p1: 0.10, p2: 0.14)
[ 120/1500] Hybrid | Tr (Loss: 1305.64, MSE: 256.6, -NLL: 22.62) | Val (MSE:
57.6, ANEES: 14.2) | Diag ( _pos: ±9.1m, p1: 0.10, p2: 0.13)
[ 130/1500] Hybrid | Tr (Loss: 2195.09, MSE: 435.8, -NLL: 16.20) | Val (MSE:
42.4, ANEES: 16.6) | Diag ( _pos: ±8.9m, p1: 0.10, p2: 0.13)
[ 140/1500] Hybrid | Tr (Loss: 1393.58, MSE: 273.6, -NLL: 25.58) | Val (MSE:
```

78.7, ANEES: 14.3) | Diag (_pos: ±9.6m, p1: 0.10, p2: 0.13)
[150/1500] Hybrid | Tr (Loss: 301.40, MSE: 57.2, -NLL: 15.20) | Val (MSE:
44.6, ANEES: 14.9) | Diag (_pos: ±8.5m, p1: 0.10, p2: 0.13)
[160/1500] Hybrid | Tr (Loss: 3051.42, MSE: 606.6, -NLL: 18.30) | Val (MSE:
42.3, ANEES: 14.5) | Diag (_pos: ±8.6m, p1: 0.10, p2: 0.13)
[170/1500] Hybrid | Tr (Loss: 4379.03, MSE: 871.3, -NLL: 22.75) | Val (MSE:
97.3, ANEES: 14.6) | Diag (_pos: ±10.1m, p1: 0.10, p2: 0.13)
[180/1500] Hybrid | Tr (Loss: 351.01, MSE: 67.1, -NLL: 15.60) | Val (MSE:
101.1, ANEES: 15.1) | Diag (_pos: ±10.2m, p1: 0.10, p2: 0.13)
[190/1500] Hybrid | Tr (Loss: 1208.71, MSE: 236.7, -NLL: 24.99) | Val (MSE:
42.5, ANEES: 14.7) | Diag (_pos: ±8.6m, p1: 0.10, p2: 0.13)
[200/1500] Hybrid | Tr (Loss: 920.79, MSE: 180.4, -NLL: 19.04) | Val (MSE:
42.2, ANEES: 14.7) | Diag (_pos: ±8.5m, p1: 0.09, p2: 0.13)
[210/1500] Hybrid | Tr (Loss: 354.81, MSE: 67.8, -NLL: 15.67) | Val (MSE:
215.3, ANEES: 14.7) | Diag (_pos: ±10.3m, p1: 0.09, p2: 0.13)
[220/1500] Hybrid | Tr (Loss: 410.37, MSE: 78.8, -NLL: 16.45) | Val (MSE:
89.0, ANEES: 14.9) | Diag (_pos: ±9.8m, p1: 0.09, p2: 0.13)
[230/1500] Hybrid | Tr (Loss: 635.78, MSE: 123.3, -NLL: 19.31) | Val (MSE:
41.9, ANEES: 15.3) | Diag (_pos: ±8.5m, p1: 0.09, p2: 0.13)
[240/1500] Hybrid | Tr (Loss: 598.59, MSE: 116.0, -NLL: 18.69) | Val (MSE:
41.6, ANEES: 15.1) | Diag (_pos: ±8.5m, p1: 0.09, p2: 0.13)
[250/1500] Hybrid | Tr (Loss: 357.74, MSE: 68.4, -NLL: 15.55) | Val (MSE:
41.9, ANEES: 15.3) | Diag (_pos: ±8.4m, p1: 0.09, p2: 0.13)
[260/1500] Hybrid | Tr (Loss: 473.88, MSE: 91.3, -NLL: 17.51) | Val (MSE:
81.3, ANEES: 15.5) | Diag (_pos: ±9.2m, p1: 0.09, p2: 0.13)
[270/1500] Hybrid | Tr (Loss: 841.63, MSE: 164.5, -NLL: 19.13) | Val (MSE:
41.7, ANEES: 15.4) | Diag (_pos: ±8.3m, p1: 0.09, p2: 0.13)
[280/1500] Hybrid | Tr (Loss: 574.43, MSE: 111.0, -NLL: 19.18) | Val (MSE:
43.0, ANEES: 15.6) | Diag (_pos: ±8.3m, p1: 0.09, p2: 0.13)
[290/1500] Hybrid | Tr (Loss: 3507.85, MSE: 697.7, -NLL: 19.55) | Val (MSE:
42.8, ANEES: 16.0) | Diag (_pos: ±8.2m, p1: 0.09, p2: 0.13)
[300/1500] Hybrid | Tr (Loss: 497.34, MSE: 96.0, -NLL: 17.43) | Val (MSE:
43.0, ANEES: 15.8) | Diag (_pos: ±8.2m, p1: 0.09, p2: 0.12)
[310/1500] Hybrid | Tr (Loss: 3039.65, MSE: 603.1, -NLL: 24.35) | Val (MSE:
42.7, ANEES: 18.5) | Diag (_pos: ±8.3m, p1: 0.09, p2: 0.12)
[320/1500] Hybrid | Tr (Loss: 532.35, MSE: 102.9, -NLL: 18.00) | Val (MSE:
81.2, ANEES: 15.6) | Diag (_pos: ±9.2m, p1: 0.09, p2: 0.12)
[330/1500] Hybrid | Tr (Loss: 452.79, MSE: 87.2, -NLL: 16.61) | Val (MSE:
43.5, ANEES: 15.7) | Diag (_pos: ±8.3m, p1: 0.09, p2: 0.12)
[340/1500] Hybrid | Tr (Loss: 2429.63, MSE: 480.8, -NLL: 25.40) | Val (MSE:
81.2, ANEES: 15.9) | Diag (_pos: ±8.9m, p1: 0.09, p2: 0.12)
[350/1500] Hybrid | Tr (Loss: 375.69, MSE: 72.1, -NLL: 14.96) | Val (MSE:
43.0, ANEES: 18.5) | Diag (_pos: ±8.2m, p1: 0.09, p2: 0.12)
[360/1500] Hybrid | Tr (Loss: 2158.73, MSE: 427.5, -NLL: 20.99) | Val (MSE:
41.6, ANEES: 15.8) | Diag (_pos: ±8.3m, p1: 0.09, p2: 0.12)
[370/1500] Hybrid | Tr (Loss: 1041.52, MSE: 204.2, -NLL: 20.51) | Val (MSE:
41.2, ANEES: 16.1) | Diag (_pos: ±8.2m, p1: 0.09, p2: 0.12)
[380/1500] Hybrid | Tr (Loss: 389.08, MSE: 74.6, -NLL: 16.27) | Val (MSE:

```
76.1, ANEES: 16.1) | Diag ( _pos: ±9.2m, p1: 0.09, p2: 0.12)
[ 390/1500] Hybrid | Tr (Loss: 2296.64, MSE: 455.8, -NLL: 17.66) | Val (MSE:
42.0, ANEES: 15.8) | Diag ( _pos: ±8.3m, p1: 0.09, p2: 0.12)
[ 400/1500] Hybrid | Tr (Loss: 569.92, MSE: 110.3, -NLL: 18.38) | Val (MSE:
88.0, ANEES: 16.6) | Diag ( _pos: ±9.3m, p1: 0.09, p2: 0.12)
[ 410/1500] Hybrid | Tr (Loss: 371.49, MSE: 71.3, -NLL: 15.01) | Val (MSE:
41.7, ANEES: 21.5) | Diag ( _pos: ±8.2m, p1: 0.09, p2: 0.12)
[ 420/1500] Hybrid | Tr (Loss: 203.82, MSE: 38.2, -NLL: 12.64) | Val (MSE:
45.7, ANEES: 16.3) | Diag ( _pos: ±8.3m, p1: 0.09, p2: 0.12)
[ 430/1500] Hybrid | Tr (Loss: 393.01, MSE: 75.4, -NLL: 16.05) | Val (MSE:
80.1, ANEES: 16.6) | Diag ( _pos: ±9.0m, p1: 0.09, p2: 0.12)
[ 440/1500] Hybrid | Tr (Loss: 3861.88, MSE: 766.5, -NLL: 29.32) | Val (MSE:
43.7, ANEES: 18.8) | Diag ( _pos: ±8.2m, p1: 0.09, p2: 0.12)
[ 450/1500] Hybrid | Tr (Loss: 221.34, MSE: 41.8, -NLL: 12.55) | Val (MSE:
41.9, ANEES: 16.9) | Diag ( _pos: ±8.1m, p1: 0.09, p2: 0.12)
[ 460/1500] Hybrid | Tr (Loss: 742.47, MSE: 145.0, -NLL: 17.40) | Val (MSE:
187.9, ANEES: 16.8) | Diag ( _pos: ±9.2m, p1: 0.09, p2: 0.12)
[ 470/1500] Hybrid | Tr (Loss: 1206.10, MSE: 236.8, -NLL: 22.09) | Val (MSE:
42.9, ANEES: 19.2) | Diag ( _pos: ±8.1m, p1: 0.09, p2: 0.12)
[ 480/1500] Hybrid | Tr (Loss: 1028.82, MSE: 201.7, -NLL: 20.26) | Val (MSE:
40.9, ANEES: 16.5) | Diag ( _pos: ±8.1m, p1: 0.09, p2: 0.12)
[ 490/1500] Hybrid | Tr (Loss: 948.57, MSE: 185.8, -NLL: 19.54) | Val (MSE:
41.1, ANEES: 16.4) | Diag ( _pos: ±8.1m, p1: 0.09, p2: 0.12)
[ 500/1500] Hybrid | Tr (Loss: 277.77, MSE: 52.8, -NLL: 13.70) | Val (MSE:
42.4, ANEES: 16.9) | Diag ( _pos: ±8.0m, p1: 0.08, p2: 0.12)
[ 510/1500] Hybrid | Tr (Loss: 5332.75, MSE: 1061.2, -NLL: 26.93) | Val (MSE:
74.9, ANEES: 17.1) | Diag ( _pos: ±8.9m, p1: 0.08, p2: 0.12)
[ 520/1500] Hybrid | Tr (Loss: 3659.63, MSE: 728.0, -NLL: 19.87) | Val (MSE:
48.1, ANEES: 16.7) | Diag ( _pos: ±8.3m, p1: 0.08, p2: 0.12)
[ 530/1500] Hybrid | Tr (Loss: 698.59, MSE: 136.6, -NLL: 15.52) | Val (MSE:
43.0, ANEES: 17.1) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 540/1500] Hybrid | Tr (Loss: 781.69, MSE: 153.1, -NLL: 16.34) | Val (MSE:
42.6, ANEES: 22.1) | Diag ( _pos: ±8.0m, p1: 0.08, p2: 0.12)
[ 550/1500] Hybrid | Tr (Loss: 1172.74, MSE: 231.1, -NLL: 17.17) | Val (MSE:
89.5, ANEES: 17.4) | Diag ( _pos: ±9.2m, p1: 0.08, p2: 0.12)
[ 560/1500] Hybrid | Tr (Loss: 1270.69, MSE: 250.6, -NLL: 17.56) | Val (MSE:
65.5, ANEES: 17.0) | Diag ( _pos: ±8.7m, p1: 0.08, p2: 0.12)
[ 570/1500] Hybrid | Tr (Loss: 282.67, MSE: 53.8, -NLL: 13.68) | Val (MSE:
42.3, ANEES: 17.1) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 580/1500] Hybrid | Tr (Loss: 1130.55, MSE: 221.9, -NLL: 21.06) | Val (MSE:
42.2, ANEES: 17.1) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 590/1500] Hybrid | Tr (Loss: 542.67, MSE: 105.0, -NLL: 17.75) | Val (MSE:
41.5, ANEES: 19.7) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 600/1500] Hybrid | Tr (Loss: 4012.20, MSE: 798.4, -NLL: 20.32) | Val (MSE:
41.5, ANEES: 17.2) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 610/1500] Hybrid | Tr (Loss: 855.28, MSE: 167.2, -NLL: 19.27) | Val (MSE:
42.2, ANEES: 17.4) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 620/1500] Hybrid | Tr (Loss: 216.18, MSE: 40.7, -NLL: 12.57) | Val (MSE:
```

```
40.8, ANEES: 17.1) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 630/1500] Hybrid | Tr (Loss: 783.59, MSE: 153.2, -NLL: 17.36) | Val (MSE:
41.7, ANEES: 17.6) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 640/1500] Hybrid | Tr (Loss: 819.93, MSE: 160.5, -NLL: 17.51) | Val (MSE:
56.1, ANEES: 17.3) | Diag ( _pos: ±8.4m, p1: 0.08, p2: 0.12)
[ 650/1500] Hybrid | Tr (Loss: 1438.08, MSE: 284.3, -NLL: 16.65) | Val (MSE:
89.5, ANEES: 17.4) | Diag ( _pos: ±9.1m, p1: 0.08, p2: 0.12)
[ 660/1500] Hybrid | Tr (Loss: 2314.99, MSE: 458.1, -NLL: 24.66) | Val (MSE:
41.2, ANEES: 17.5) | Diag ( _pos: ±8.0m, p1: 0.08, p2: 0.12)
[ 670/1500] Hybrid | Tr (Loss: 471.65, MSE: 90.9, -NLL: 17.11) | Val (MSE:
44.3, ANEES: 17.5) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 680/1500] Hybrid | Tr (Loss: 4876.25, MSE: 970.1, -NLL: 25.83) | Val (MSE:
74.9, ANEES: 22.7) | Diag ( _pos: ±8.7m, p1: 0.08, p2: 0.12)
[ 690/1500] Hybrid | Tr (Loss: 428.51, MSE: 82.5, -NLL: 15.79) | Val (MSE:
41.7, ANEES: 17.8) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 700/1500] Hybrid | Tr (Loss: 220.54, MSE: 41.6, -NLL: 12.54) | Val (MSE:
40.8, ANEES: 17.9) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 710/1500] Hybrid | Tr (Loss: 350.95, MSE: 67.3, -NLL: 14.35) | Val (MSE:
40.9, ANEES: 17.5) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 720/1500] Hybrid | Tr (Loss: 370.14, MSE: 71.0, -NLL: 15.14) | Val (MSE:
116.0, ANEES: 17.8) | Diag ( _pos: ±9.5m, p1: 0.08, p2: 0.12)
[ 730/1500] Hybrid | Tr (Loss: 2823.46, MSE: 561.7, -NLL: 15.15) | Val (MSE:
42.6, ANEES: 17.6) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 740/1500] Hybrid | Tr (Loss: 4017.79, MSE: 798.8, -NLL: 23.63) | Val (MSE:
41.7, ANEES: 17.4) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 750/1500] Hybrid | Tr (Loss: 2897.09, MSE: 574.8, -NLL: 22.98) | Val (MSE:
92.6, ANEES: 17.8) | Diag ( _pos: ±9.1m, p1: 0.08, p2: 0.12)
[ 760/1500] Hybrid | Tr (Loss: 393.17, MSE: 75.5, -NLL: 15.52) | Val (MSE:
41.1, ANEES: 17.4) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 770/1500] Hybrid | Tr (Loss: 221.93, MSE: 41.9, -NLL: 12.44) | Val (MSE:
40.7, ANEES: 17.2) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 780/1500] Hybrid | Tr (Loss: 913.05, MSE: 179.0, -NLL: 18.05) | Val (MSE:
42.2, ANEES: 17.5) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 790/1500] Hybrid | Tr (Loss: 225.05, MSE: 42.5, -NLL: 12.53) | Val (MSE:
79.2, ANEES: 20.7) | Diag ( _pos: ±8.7m, p1: 0.08, p2: 0.12)
[ 800/1500] Hybrid | Tr (Loss: 1094.74, MSE: 215.7, -NLL: 16.25) | Val (MSE:
41.9, ANEES: 17.6) | Diag ( _pos: ±8.0m, p1: 0.08, p2: 0.12)
[ 810/1500] Hybrid | Tr (Loss: 1177.97, MSE: 232.2, -NLL: 17.03) | Val (MSE:
41.9, ANEES: 17.7) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 820/1500] Hybrid | Tr (Loss: 1241.70, MSE: 244.6, -NLL: 18.64) | Val (MSE:
41.5, ANEES: 20.4) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 830/1500] Hybrid | Tr (Loss: 223.01, MSE: 42.1, -NLL: 12.51) | Val (MSE:
42.7, ANEES: 17.9) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 840/1500] Hybrid | Tr (Loss: 2312.61, MSE: 459.1, -NLL: 16.95) | Val (MSE:
42.8, ANEES: 20.6) | Diag ( _pos: ±7.7m, p1: 0.08, p2: 0.11)
[ 850/1500] Hybrid | Tr (Loss: 413.50, MSE: 79.5, -NLL: 15.79) | Val (MSE:
46.0, ANEES: 17.6) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.11)
[ 860/1500] Hybrid | Tr (Loss: 2619.08, MSE: 519.7, -NLL: 20.49) | Val (MSE:
```

```
43.1, ANEES: 18.3) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 870/1500] Hybrid | Tr (Loss: 1315.98, MSE: 259.6, -NLL: 17.90) | Val (MSE:
42.0, ANEES: 17.7) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 880/1500] Hybrid | Tr (Loss: 748.69, MSE: 146.4, -NLL: 16.59) | Val (MSE:
41.6, ANEES: 17.5) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 890/1500] Hybrid | Tr (Loss: 2772.38, MSE: 550.7, -NLL: 19.07) | Val (MSE:
52.7, ANEES: 17.7) | Diag ( _pos: ±8.3m, p1: 0.08, p2: 0.12)
[ 900/1500] Hybrid | Tr (Loss: 448.26, MSE: 86.4, -NLL: 16.20) | Val (MSE:
44.3, ANEES: 18.2) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 910/1500] Hybrid | Tr (Loss: 369.44, MSE: 70.9, -NLL: 15.00) | Val (MSE:
41.3, ANEES: 17.5) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 920/1500] Hybrid | Tr (Loss: 350.79, MSE: 67.2, -NLL: 14.86) | Val (MSE:
42.2, ANEES: 18.4) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 930/1500] Hybrid | Tr (Loss: 2628.00, MSE: 521.1, -NLL: 22.44) | Val (MSE:
41.3, ANEES: 20.4) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 940/1500] Hybrid | Tr (Loss: 523.51, MSE: 101.2, -NLL: 17.34) | Val (MSE:
78.0, ANEES: 17.8) | Diag ( _pos: ±8.8m, p1: 0.08, p2: 0.12)
[ 950/1500] Hybrid | Tr (Loss: 2992.71, MSE: 595.0, -NLL: 17.69) | Val (MSE:
41.4, ANEES: 20.3) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[ 960/1500] Hybrid | Tr (Loss: 213.66, MSE: 40.3, -NLL: 12.22) | Val (MSE:
42.6, ANEES: 17.8) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 970/1500] Hybrid | Tr (Loss: 2405.13, MSE: 477.4, -NLL: 18.24) | Val (MSE:
39.3, ANEES: 20.6) | Diag ( _pos: ±7.7m, p1: 0.08, p2: 0.12)
[ 980/1500] Hybrid | Tr (Loss: 950.11, MSE: 185.9, -NLL: 20.36) | Val (MSE:
42.9, ANEES: 18.2) | Diag ( _pos: ±7.9m, p1: 0.08, p2: 0.12)
[ 990/1500] Hybrid | Tr (Loss: 2213.26, MSE: 439.4, -NLL: 16.04) | Val (MSE:
41.4, ANEES: 20.3) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[1000/1500] Hybrid | Tr (Loss: 347.24, MSE: 66.6, -NLL: 14.40) | Val (MSE:
42.4, ANEES: 17.9) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.12)
[1010/1500] Hybrid | Tr (Loss: 725.64, MSE: 141.1, -NLL: 20.12) | Val (MSE:
75.1, ANEES: 18.4) | Diag ( _pos: ±8.7m, p1: 0.08, p2: 0.12)
[1020/1500] Hybrid | Tr (Loss: 1201.84, MSE: 236.9, -NLL: 17.32) | Val (MSE:
53.7, ANEES: 18.4) | Diag ( _pos: ±8.1m, p1: 0.08, p2: 0.11)
[1030/1500] Hybrid | Tr (Loss: 424.40, MSE: 82.2, -NLL: 13.63) | Val (MSE:
41.9, ANEES: 18.1) | Diag ( _pos: ±7.8m, p1: 0.08, p2: 0.11)
[1040/1500] Hybrid | Tr (Loss: 877.54, MSE: 171.9, -NLL: 18.13) | Val (MSE:
42.2, ANEES: 18.7) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1050/1500] Hybrid | Tr (Loss: 2615.12, MSE: 518.9, -NLL: 20.52) | Val (MSE:
41.3, ANEES: 18.2) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1060/1500] Hybrid | Tr (Loss: 742.51, MSE: 145.1, -NLL: 16.87) | Val (MSE:
41.1, ANEES: 18.4) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1070/1500] Hybrid | Tr (Loss: 351.79, MSE: 67.5, -NLL: 14.40) | Val (MSE:
41.8, ANEES: 18.2) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1080/1500] Hybrid | Tr (Loss: 212.54, MSE: 40.1, -NLL: 12.11) | Val (MSE:
40.1, ANEES: 18.1) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1090/1500] Hybrid | Tr (Loss: 3853.63, MSE: 766.1, -NLL: 23.02) | Val (MSE:
40.9, ANEES: 20.8) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.12)
[1100/1500] Hybrid | Tr (Loss: 2262.53, MSE: 448.3, -NLL: 20.90) | Val (MSE:
```

42.4, ANEES: 18.8) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1110/1500] Hybrid | Tr (Loss: 611.34, MSE: 119.3, -NLL: 14.64) | Val (MSE:
57.5, ANEES: 18.2) | Diag (_pos: ±8.2m, p1: 0.07, p2: 0.12)
[1120/1500] Hybrid | Tr (Loss: 305.20, MSE: 58.4, -NLL: 13.45) | Val (MSE:
41.3, ANEES: 18.0) | Diag (_pos: ±7.8m, p1: 0.07, p2: 0.12)
[1130/1500] Hybrid | Tr (Loss: 1294.09, MSE: 255.3, -NLL: 17.58) | Val (MSE:
74.8, ANEES: 20.4) | Diag (_pos: ±8.7m, p1: 0.07, p2: 0.12)
[1140/1500] Hybrid | Tr (Loss: 5098.96, MSE: 1014.7, -NLL: 25.68) | Val (MSE:
77.2, ANEES: 18.4) | Diag (_pos: ±8.6m, p1: 0.07, p2: 0.12)
[1150/1500] Hybrid | Tr (Loss: 213.65, MSE: 40.3, -NLL: 12.06) | Val (MSE:
57.7, ANEES: 18.5) | Diag (_pos: ±8.1m, p1: 0.07, p2: 0.12)
[1160/1500] Hybrid | Tr (Loss: 491.33, MSE: 94.9, -NLL: 16.64) | Val (MSE:
41.2, ANEES: 18.3) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1170/1500] Hybrid | Tr (Loss: 221.87, MSE: 41.9, -NLL: 12.48) | Val (MSE:
41.7, ANEES: 20.7) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1180/1500] Hybrid | Tr (Loss: 1768.44, MSE: 349.7, -NLL: 19.97) | Val (MSE:
39.9, ANEES: 18.1) | Diag (_pos: ±7.6m, p1: 0.07, p2: 0.12)
[1190/1500] Hybrid | Tr (Loss: 204.77, MSE: 38.6, -NLL: 11.94) | Val (MSE:
41.3, ANEES: 21.0) | Diag (_pos: ±7.6m, p1: 0.07, p2: 0.12)
[1200/1500] Hybrid | Tr (Loss: 434.21, MSE: 83.9, -NLL: 14.69) | Val (MSE:
87.9, ANEES: 18.3) | Diag (_pos: ±8.7m, p1: 0.07, p2: 0.12)
[1210/1500] Hybrid | Tr (Loss: 4655.96, MSE: 925.8, -NLL: 26.82) | Val (MSE:
63.5, ANEES: 18.1) | Diag (_pos: ±8.4m, p1: 0.07, p2: 0.12)
[1220/1500] Hybrid | Tr (Loss: 350.03, MSE: 67.0, -NLL: 15.13) | Val (MSE:
48.3, ANEES: 18.3) | Diag (_pos: ±8.1m, p1: 0.07, p2: 0.12)
[1230/1500] Hybrid | Tr (Loss: 1593.99, MSE: 315.0, -NLL: 18.89) | Val (MSE:
42.7, ANEES: 18.8) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1240/1500] Hybrid | Tr (Loss: 1615.29, MSE: 319.7, -NLL: 17.03) | Val (MSE:
42.6, ANEES: 18.2) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1250/1500] Hybrid | Tr (Loss: 882.60, MSE: 172.7, -NLL: 19.28) | Val (MSE:
86.4, ANEES: 20.7) | Diag (_pos: ±8.9m, p1: 0.07, p2: 0.12)
[1260/1500] Hybrid | Tr (Loss: 2671.58, MSE: 530.7, -NLL: 18.19) | Val (MSE:
50.4, ANEES: 18.6) | Diag (_pos: ±8.0m, p1: 0.07, p2: 0.12)
[1270/1500] Hybrid | Tr (Loss: 234.58, MSE: 44.4, -NLL: 12.64) | Val (MSE:
41.2, ANEES: 18.1) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.12)
[1280/1500] Hybrid | Tr (Loss: 206.74, MSE: 38.9, -NLL: 12.14) | Val (MSE:
40.1, ANEES: 18.3) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.11)
[1290/1500] Hybrid | Tr (Loss: 1998.97, MSE: 395.1, -NLL: 23.65) | Val (MSE:
41.7, ANEES: 19.1) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.11)
[1300/1500] Hybrid | Tr (Loss: 408.79, MSE: 78.7, -NLL: 15.40) | Val (MSE:
41.0, ANEES: 18.7) | Diag (_pos: ±7.6m, p1: 0.07, p2: 0.11)
[1310/1500] Hybrid | Tr (Loss: 1177.70, MSE: 232.2, -NLL: 16.92) | Val (MSE:
53.5, ANEES: 18.6) | Diag (_pos: ±8.0m, p1: 0.07, p2: 0.11)
[1320/1500] Hybrid | Tr (Loss: 381.77, MSE: 73.3, -NLL: 15.13) | Val (MSE:
54.4, ANEES: 18.9) | Diag (_pos: ±8.0m, p1: 0.07, p2: 0.11)
[1330/1500] Hybrid | Tr (Loss: 1580.08, MSE: 312.4, -NLL: 18.18) | Val (MSE:
41.0, ANEES: 21.0) | Diag (_pos: ±7.7m, p1: 0.07, p2: 0.11)
[1340/1500] Hybrid | Tr (Loss: 781.92, MSE: 153.1, -NLL: 16.47) | Val (MSE:

```
81.6, ANEES: 18.8) | Diag ( _pos: ±8.5m, p1: 0.07, p2: 0.11)
[1350/1500] Hybrid | Tr (Loss: 357.77, MSE: 68.7, -NLL: 14.12) | Val (MSE:
41.4, ANEES: 21.2) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
[1360/1500] Hybrid | Tr (Loss: 439.88, MSE: 84.8, -NLL: 15.97) | Val (MSE:
43.1, ANEES: 18.9) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1370/1500] Hybrid | Tr (Loss: 259.50, MSE: 49.2, -NLL: 13.55) | Val (MSE:
40.2, ANEES: 18.6) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
[1380/1500] Hybrid | Tr (Loss: 370.96, MSE: 71.3, -NLL: 14.38) | Val (MSE:
144.2, ANEES: 18.6) | Diag ( _pos: ±10.1m, p1: 0.07, p2: 0.12)
[1390/1500] Hybrid | Tr (Loss: 2225.06, MSE: 440.8, -NLL: 20.95) | Val (MSE:
41.1, ANEES: 18.5) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.12)
[1400/1500] Hybrid | Tr (Loss: 758.91, MSE: 147.9, -NLL: 19.37) | Val (MSE:
42.4, ANEES: 23.6) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.12)
[1410/1500] Hybrid | Tr (Loss: 216.61, MSE: 40.9, -NLL: 12.16) | Val (MSE:
42.0, ANEES: 18.6) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.12)
[1420/1500] Hybrid | Tr (Loss: 401.70, MSE: 77.4, -NLL: 14.90) | Val (MSE:
41.5, ANEES: 18.9) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.12)
[1430/1500] Hybrid | Tr (Loss: 2043.77, MSE: 404.8, -NLL: 19.72) | Val (MSE:
40.9, ANEES: 18.8) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
[1440/1500] Hybrid | Tr (Loss: 2645.22, MSE: 524.9, -NLL: 20.50) | Val (MSE:
77.4, ANEES: 18.6) | Diag ( _pos: ±8.6m, p1: 0.07, p2: 0.11)
[1450/1500] Hybrid | Tr (Loss: 3373.50, MSE: 670.4, -NLL: 21.44) | Val (MSE:
40.9, ANEES: 21.1) | Diag ( _pos: ±7.7m, p1: 0.07, p2: 0.11)
[1460/1500] Hybrid | Tr (Loss: 412.36, MSE: 79.6, -NLL: 14.16) | Val (MSE:
41.6, ANEES: 18.6) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
[1470/1500] Hybrid | Tr (Loss: 213.33, MSE: 40.3, -NLL: 11.96) | Val (MSE:
120.5, ANEES: 18.7) | Diag ( _pos: ±9.5m, p1: 0.07, p2: 0.11)
[1480/1500] Hybrid | Tr (Loss: 3246.27, MSE: 644.8, -NLL: 22.13) | Val (MSE:
58.5, ANEES: 18.7) | Diag ( _pos: ±8.0m, p1: 0.07, p2: 0.11)
[1490/1500] Hybrid | Tr (Loss: 207.73, MSE: 39.1, -NLL: 11.99) | Val (MSE:
39.9, ANEES: 18.5) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
[1500/1500] Hybrid | Tr (Loss: 499.82, MSE: 96.7, -NLL: 16.48) | Val (MSE:
41.3, ANEES: 18.6) | Diag ( _pos: ±7.6m, p1: 0.07, p2: 0.11)
--------------------------------------------------------------------------------
--------------------
 Trénink dokončen. Načítám nejlepší model z iterace 20 (Val MSE: 43.0453, Val
ANEES: 13.1284)
 Fáze 4 dokončena. Model uložen.

 CELÝ TRÉNINK DOKONČEN.
```

[15]:
```python
if False:
    # save model.
    save_path = f'best_mse_and_anees_bknet.pth'
    torch.save(state_knet2.state_dict(), save_path)
    print(f"Model saved to '{save_path}'.")
```

[16]:
```
# import torch
# from state_NN_models import TAN # Tvůj import

# # 1. Cesta k uloženému modelu z konce Fáze 4 (zkontroluj si přesný název v␣
 ↪tvé složce)
# checkpoint_path = "checkpoints/bkn_phase_4_len100.pth"

# # 2. Inicializace prázdného modelu (musí mít naprosto shodné parametry jako v␣
 ↪tréninku!)
# best_bkn_model = TAN.StateBayesianKalmanNetTAN(
#     system_model=system_model,
#     device=device,
#     hidden_size_multiplier=8,
#     output_layer_multiplier=4,
#     num_gru_layers=1,
#     init_max_dropout=0.5,
#     init_min_dropout=0.3
# ).to(device)

# # 3. Nahrání natrénovaných vah do modelu
# # (map_location zajistí, že se to správně nahraje na CPU nebo GPU podle toho,␣
 ↪co zrovna používáš)
# best_bkn_model.load_state_dict(torch.load(checkpoint_path,␣
 ↪map_location=device))
# state_knet2=best_bkn_model
# # 4. Přepnutí modelu do evaluačního (testovacího) režimu
# best_bkn_model.eval()

# print(" Nejlepší BKN model úspěšně načten a je připraven k testování!")
```

1 Test na synteticke trajektorii

[20]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import Filters
import os
from tqdm import tqdm
from Filters import TAN

# === KONFIGURACE ===
TEST_DATA_PATH = './generated_data_synthetic_controlled/test_set/test.pt'
PLOT_PER_ITERATION = True  # Vykreslovat graf pro každou trajektorii?
MAX_TEST_SAMPLES = 20        # Kolik trajektorií z test setu vyhodnotit
J_EVALUATION = 100           # Počet Monte Carlo vzorků pro BKN (Ensemble size)
```

```
print(f"=== VYHODNOCENÍ BKN NA TESTOVACÍ SADĚ (s ANEES) ===")
print(f"Načítám data z: {TEST_DATA_PATH}")

# 1. Načtení Testovací sady
if not os.path.exists(TEST_DATA_PATH):
    raise FileNotFoundError(f"Soubor {TEST_DATA_PATH} neexistuje!")

# Předpokládáme, že 'device' je definováno
if 'device' not in globals():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

test_data = torch.load(TEST_DATA_PATH, map_location=device)
X_test_all = test_data['x']  # Ground Truth [N, Seq, 4]
Y_test_all = test_data['y']  # Measurements [N, Seq, 3]

n_samples = min(X_test_all.shape[0], MAX_TEST_SAMPLES)
print(f"Počet testovacích trajektorií: {n_samples}")
print(f"Ensemble size (BKN): {J_EVALUATION}")
print(f"Délka sekvence: {X_test_all.shape[1]}")
print("Modely: BKN vs. UKF vs. PF vs. APF")

# 2. Inicializace pro sběr dat
detailed_results = []
trajectory_history = []
agg_mse = {"BKN": [], "UKF": [], "PF": [], "APF": []}
agg_pos = {"BKN": [], "UKF": [], "PF": [], "APF": []}
agg_anees = {"BKN": [], "UKF": [], "PF": [], "APF": []} # Nový list pro ANEES

# Ujistíme se, že BKN je v eval módu
state_knet2.eval()

# --- POMOCNÁ FUNKCE PRO ANEES ---
def calculate_anees(gt, est, P):
    """
    Vypočítá Average Normalized Estimation Error Squared.
    gt: Ground Truth [T, Dim] (NumPy)
    est: Odhad [T, Dim] (NumPy)
    P: Kovariační matice [T, Dim, Dim] (NumPy)
    """
    T = min(len(gt), len(est), len(P))
    anees_vals = []

    # Oříznutí na stejnou délku
    gt = gt[:T]
    est = est[:T]
    P = P[:T]
```

```
    for t in range(T):
        e_t = gt[t] - est[t] # Chyba v čase t
        P_t = P[t]

        try:
            # Inverze kovariance
            # Přičteme malé epsilon na diagonálu pro numerickou stabilitu, pokud je singulární
            if np.linalg.cond(P_t) > 1e10:
                P_t = P_t + np.eye(P_t.shape[0]) * 1e-6

            P_inv = np.linalg.inv(P_t)

            # Mahalanobisova vzdálenost: e^T * P^-1 * e
            anees_t = e_t.T @ P_inv @ e_t
            anees_vals.append(anees_t)
        except np.linalg.LinAlgError:
            anees_vals.append(np.nan)

    return np.nanmean(anees_vals)

# --- HLAVNÍ SMYČKA ---
for i in tqdm(range(n_samples), desc="Evaluace"):

    # A) Příprava dat
    x_gt_tensor = X_test_all[i].to(device)
    y_obs_tensor = Y_test_all[i].to(device)

    x_gt = x_gt_tensor.cpu().numpy()
    seq_len = x_gt.shape[0]
    true_init_state = x_gt_tensor[0]

    # --- B) BKN (Ensemble) ---
    with torch.no_grad():
        init_batch = true_init_state.unsqueeze(0).repeat(J_EVALUATION, 1)
        state_knet2.reset(batch_size=J_EVALUATION, initial_state=init_batch)

        bkn_preds = []
        y_input_batch = y_obs_tensor.unsqueeze(0).repeat(J_EVALUATION, 1, 1)

        for t in range(1, seq_len):
            y_t = y_input_batch[:, t, :]
            x_est, _ = state_knet2.step(y_t)
            bkn_preds.append(x_est)

        if len(bkn_preds) > 0:
```

```
            bkn_preds_tensor = torch.stack(bkn_preds, dim=1) # [J, Seq-1, 4]
            full_bkn_ensemble = torch.cat([init_batch.unsqueeze(1),␣
↪bkn_preds_tensor], dim=1) # [J, Seq, 4]

            # Mean Estimate
            x_est_mean = full_bkn_ensemble.mean(dim=0)
            x_est_bkn = x_est_mean.cpu().numpy()

            # --- VÝPOČET KOVARIANCE PRO BKN ---
            # P = 1/(J-1) * sum((x_j - x_mean) * (x_j - x_mean)^T)
            # Vycentrování
            residuals = full_bkn_ensemble - x_est_mean.unsqueeze(0) # [J, Seq,␣
↪4]
            # Permute pro batch matmul: [Seq, J, 4] a [Seq, 4, J]
            residuals = residuals.permute(1, 2, 0) # [Seq, 4, J]

            # Batch matrix multiplication: (Seq, 4, J) @ (Seq, J, 4) -> (Seq,␣
↪4, 4)
            P_bkn_tensor = torch.bmm(residuals, residuals.transpose(1, 2)) /␣
↪(J_EVALUATION - 1)
            # Přičtení process noise/stabilitu (volitelné, BKN variance je␣
↪epistemická)
            P_bkn = P_bkn_tensor.cpu().numpy()

        else:
            x_est_bkn = x_gt
            P_bkn = np.eye(4)[np.newaxis, :, :].repeat(len(x_gt), axis=0)

    # --- C) Klasické Filtry ---

    # UKF
    ukf_ideal = Filters.UnscentedKalmanFilter(system_model)
    ukf_res = ukf_ideal.process_sequence(y_seq=y_obs_tensor,␣
↪Ex0=true_init_state, P0=system_model.P0)
    x_est_ukf = ukf_res['x_filtered'].cpu().numpy()
    # Získání P pro UKF (zkusíme různé klíče)
    P_ukf = ukf_res.get('P_filtered', ukf_res.get('P', None))
    if P_ukf is not None: P_ukf = P_ukf.cpu().numpy()

    # PF
    pf = TAN.ParticleFilterTAN(system_model, num_particles=1000)
    pf_res = pf.process_sequence(y_seq=y_obs_tensor, Ex0=true_init_state,␣
↪P0=system_model.P0)
    x_est_pf = pf_res['x_filtered'].cpu().numpy()
    P_pf = pf_res.get('P_filtered', pf_res.get('P', None))
    if P_pf is not None: P_pf = P_pf.cpu().numpy()
```

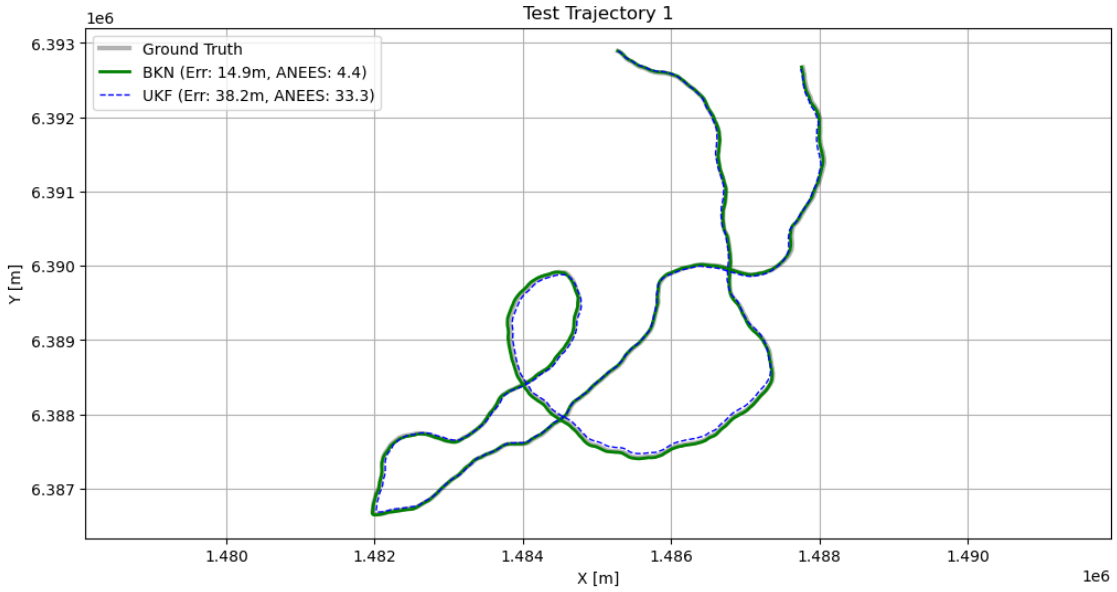
```
    # APF
    apf = TAN.AuxiliaryParticleFilterTAN(system_model, num_particles=2000)
    apf_res = apf.process_sequence(y_seq=y_obs_tensor, Ex0=true_init_state,␣
↪P0=system_model.P0)
    x_est_apf = apf_res['x_filtered'].cpu().numpy()
    P_apf = apf_res.get('P_filtered', apf_res.get('P', None))
    if P_apf is not None: P_apf = P_apf.cpu().numpy()

    # --- D) Výpočet chyb a ANEES ---
    min_len = min(len(x_gt), len(x_est_bkn), len(x_est_ukf))

    def calc_metrics(est, gt, P_mat):
        diff = est[:min_len] - gt[:min_len]
        mse = np.mean(np.sum(diff[:, :2]**2, axis=1))
        pos_err = np.mean(np.sqrt(diff[:, 0]**2 + diff[:, 1]**2))

        anees = np.nan
        if P_mat is not None:
            anees = calculate_anees(gt[:min_len], est[:min_len], P_mat[:
↪min_len])

        return mse, pos_err, anees

    # Calculate for all
    mse_bkn, pos_bkn, anees_bkn = calc_metrics(x_est_bkn, x_gt, P_bkn)
    mse_ukf, pos_ukf, anees_ukf = calc_metrics(x_est_ukf, x_gt, P_ukf)
    mse_pf, pos_pf, anees_pf = calc_metrics(x_est_pf, x_gt, P_pf)
    mse_apf, pos_apf, anees_apf = calc_metrics(x_est_apf, x_gt, P_apf)

    # Uložení
    agg_mse["BKN"].append(mse_bkn); agg_pos["BKN"].append(pos_bkn);␣
↪agg_anees["BKN"].append(anees_bkn)
    agg_mse["UKF"].append(mse_ukf); agg_pos["UKF"].append(pos_ukf);␣
↪agg_anees["UKF"].append(anees_ukf)
    agg_mse["PF"].append(mse_pf);   agg_pos["PF"].append(pos_pf);   ␣
↪agg_anees["PF"].append(anees_pf)
    agg_mse["APF"].append(mse_apf); agg_pos["APF"].append(pos_apf);␣
↪agg_anees["APF"].append(anees_apf)

    detailed_results.append({
        "Run_ID": i + 1,
        "BKN_PosErr": pos_bkn, "BKN_ANEES": anees_bkn,
        "UKF_PosErr": pos_ukf, "UKF_ANEES": anees_ukf,
        "PF_PosErr": pos_pf,   "PF_ANEES": anees_pf,
        "APF_PosErr": pos_apf, "APF_ANEES": anees_apf
```

```
    })

    if i < 2:  # Uložíme detailní časovou historii pro první 2 trajektorie
        trajectory_history.append({
            "gt": x_gt,
            "bkn_est": x_est_bkn,
            "bkn_P": P_bkn,
            "ukf_est": x_est_ukf,
            "ukf_P": P_ukf
        })
    # E) Vykreslení
    if PLOT_PER_ITERATION:
        fig = plt.figure(figsize=(12, 6))
        plt.plot(x_gt[:, 0], x_gt[:, 1], 'k-', linewidth=3, alpha=0.3,␣
 ↪label='Ground Truth')
        plt.plot(x_est_bkn[:, 0], x_est_bkn[:, 1], 'g-', linewidth=2,␣
 ↪label=f'BKN (Err: {pos_bkn:.1f}m, ANEES: {anees_bkn:.1f})')
        plt.plot(x_est_ukf[:, 0], x_est_ukf[:, 1], 'b--', linewidth=1,␣
 ↪label=f'UKF (Err: {pos_ukf:.1f}m, ANEES: {anees_ukf:.1f})')
        # Pro přehlednost vykreslíme jen BKN a UKF, případně odkomentujte PF/APF
        # plt.plot(x_est_pf[:, 0], x_est_pf[:, 1], 'r:', linewidth=1, alpha=0.
 ↪6, label='PF')
        plt.title(f"Test Trajectory {i+1}")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.legend()
        plt.axis('equal')
        plt.grid(True)
        plt.show()

# --- VÝPIS VÝSLEDKŮ ---
df_results = pd.DataFrame(detailed_results)
print("\n" + "="*120)
print(f"DETAILNÍ VÝSLEDKY (Pozice v metrech | ANEES - ideál ~4.0)")
print("="*120)
pd.options.display.float_format = '{:,.2f}'.format
print(df_results[["Run_ID", "BKN_PosErr", "BKN_ANEES", "UKF_PosErr",␣
 ↪"UKF_ANEES", "PF_PosErr", "APF_PosErr"]])

print("\n" + "="*120)
print(f"SOUHRNNÁ STATISTIKA ({n_samples} trajektorií)")
print("="*120)

def get_stats(key):
    return (np.nanmean(agg_mse[key]), np.nanstd(agg_mse[key]),
            np.nanmean(agg_pos[key]), np.nanstd(agg_pos[key]),
            np.nanmean(agg_anees[key]), np.nanstd(agg_anees[key]))
```

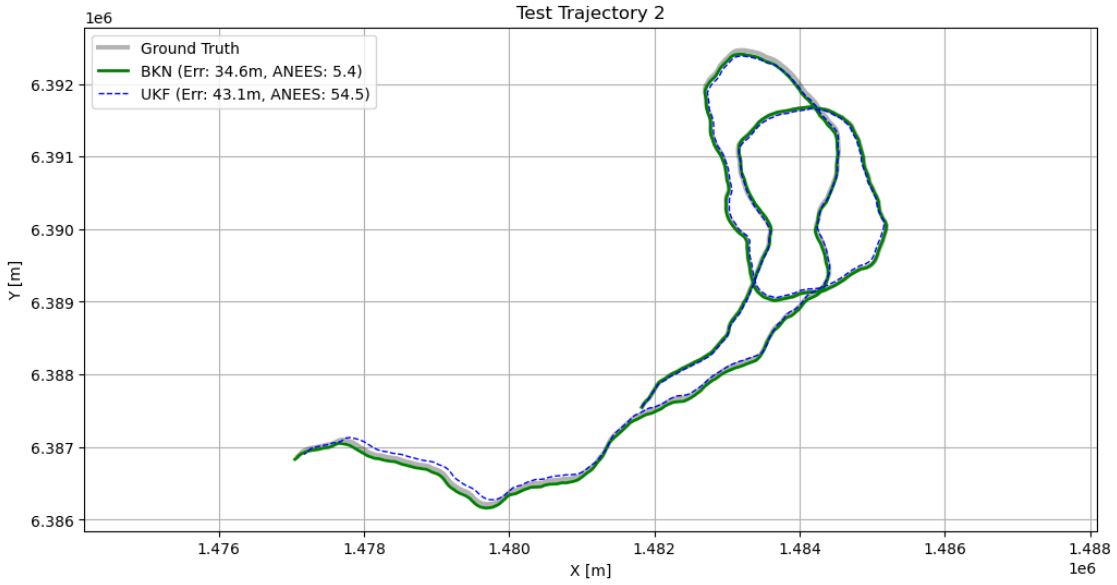
```
bkn_s = get_stats("BKN")
ukf_s = get_stats("UKF")
pf_s = get_stats("PF")
apf_s = get_stats("APF")

# Formátování tabulky
header = f"{'Model':<10} | {'Pos Error [m] (Mean ± Std)':<30} | {'ANEES (Mean ±␣
 ↪Std)':<30}"
print(header)
print("-" * len(header))
print(f"{'BKN':<10} | {bkn_s[2]:.2f} ± {bkn_s[3]:.2f} m {'':<14} | {bkn_s[4]:.
 ↪2f} ± {bkn_s[5]:.2f}")
print(f"{'UKF':<10} | {ukf_s[2]:.2f} ± {ukf_s[3]:.2f} m {'':<14} | {ukf_s[4]:.
 ↪2f} ± {ukf_s[5]:.2f}")
print(f"{'PF':<10} | {pf_s[2]:.2f} ± {pf_s[3]:.2f} m {'':<14} | {pf_s[4]:.2f} ±␣
 ↪{pf_s[5]:.2f}")
print(f"{'APF':<10} | {apf_s[2]:.2f} ± {apf_s[3]:.2f} m {'':<14} | {apf_s[4]:.
 ↪2f} ± {apf_s[5]:.2f}")
print("="*120)

# Grafické porovnání (Boxplot Position Error)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.boxplot([agg_pos["BKN"], agg_pos["UKF"], agg_pos["PF"]], labels=['BKN',␣
 ↪'UKF', 'PF'], patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title("Position Error [m]")
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

# Grafické porovnání (Boxplot ANEES)
plt.subplot(1, 2, 2)
# Filtrujeme NaN pro boxplot
anees_data = [
    [x for x in agg_anees["BKN"] if not np.isnan(x)],
    [x for x in agg_anees["UKF"] if not np.isnan(x)],
    [x for x in agg_anees["PF"] if not np.isnan(x)]
]
plt.boxplot(anees_data, labels=['BKN', 'UKF', 'PF'], patch_artist=True,␣
 ↪boxprops=dict(facecolor='lightgreen'))
plt.axhline(y=4.0, color='r', linestyle='--', label='Ideal (4.0)')
plt.title("ANEES (Consistency)")
plt.legend()
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
=== VYHODNOCENÍ BKN NA TESTOVACÍ SADĚ (s ANEES) ===
```

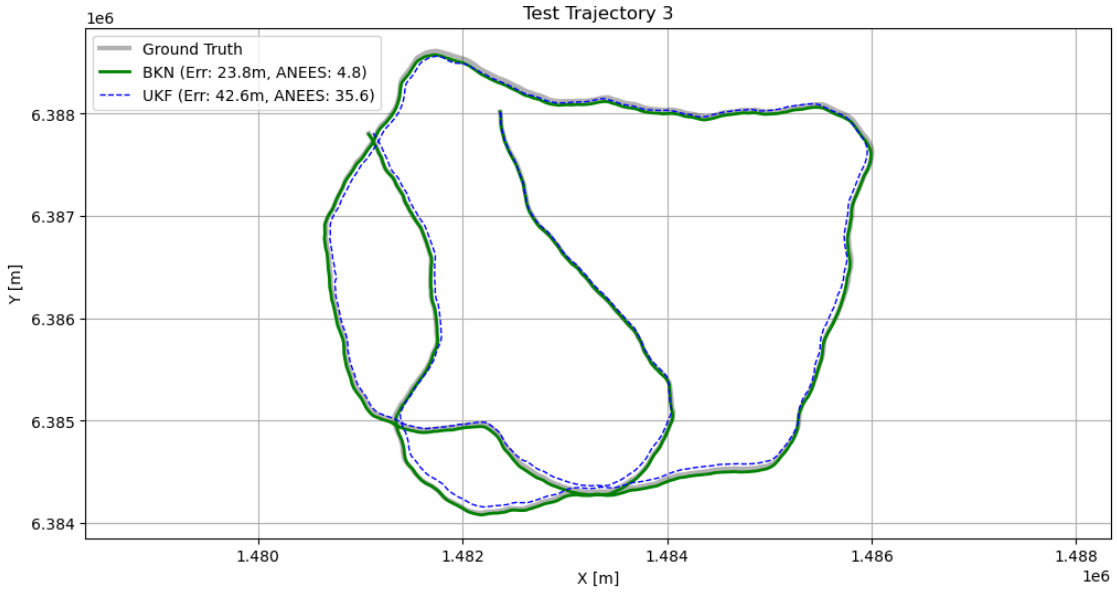
```
Načítám data z: ./generated_data_synthetic_controlled/test_set/test.pt
Počet testovacích trajektorií: 20
Ensemble size (BKN): 100
Délka sekvence: 1000
Modely: BKN vs. UKF vs. PF vs. APF
Evaluace:   0%|          | 0/20 [00:00<?, ?it/s]
```



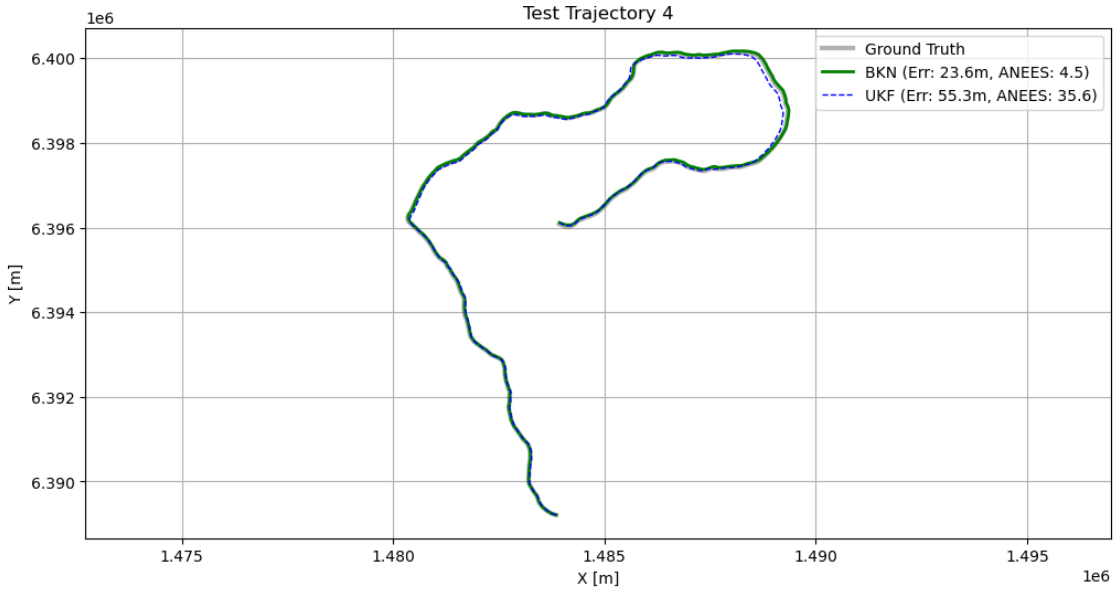

```
Evaluace:   5%|          | 1/20 [00:08<02:34,  8.15s/it]
```



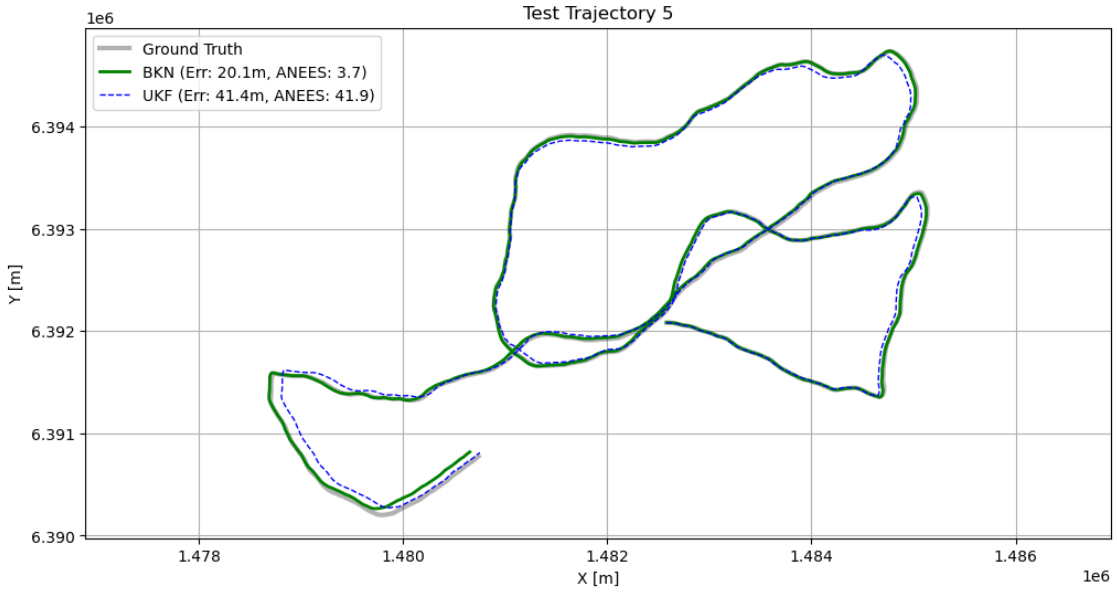
Evaluace: 10%| | 2/20 [00:16<02:24, 8.04s/it]



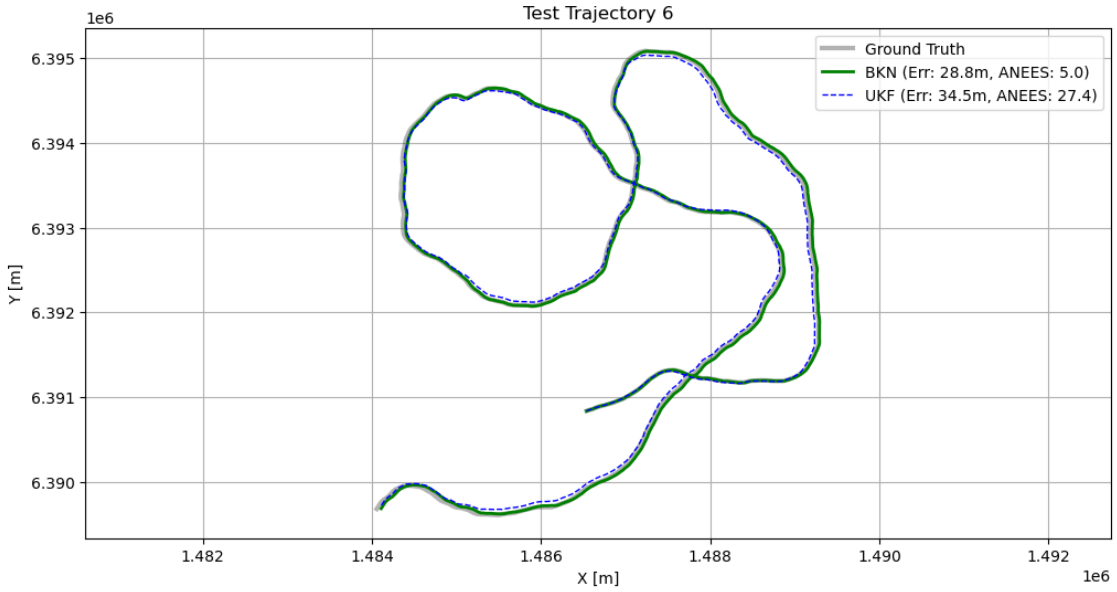
Evaluace: 15%| | 3/20 [00:24<02:16, 8.02s/it]



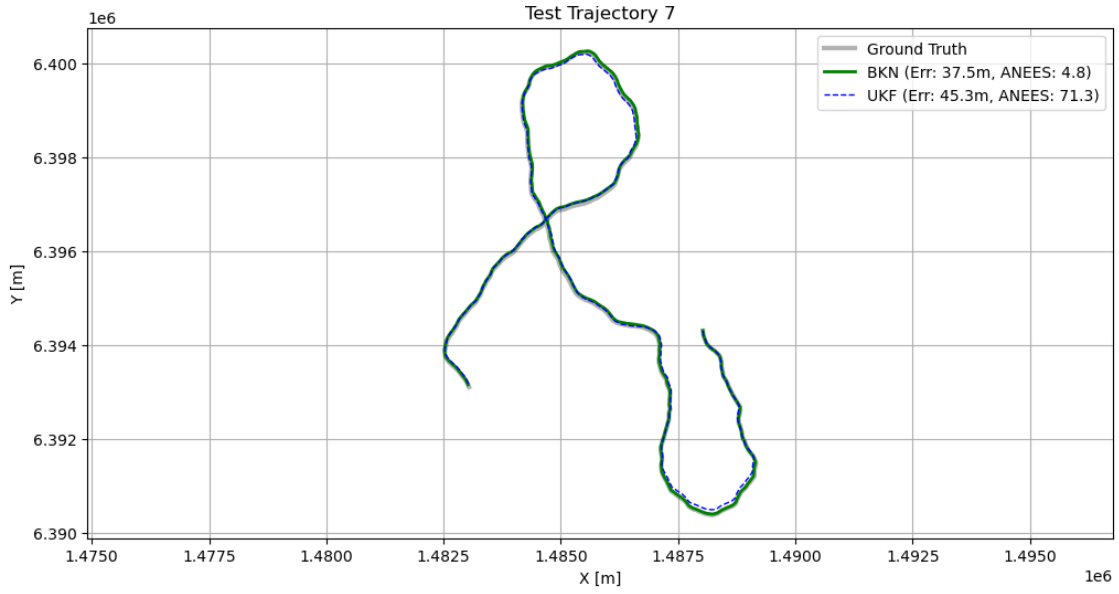
Evaluace: 20%| | 4/20 [00:32<02:07, 7.99s/it]



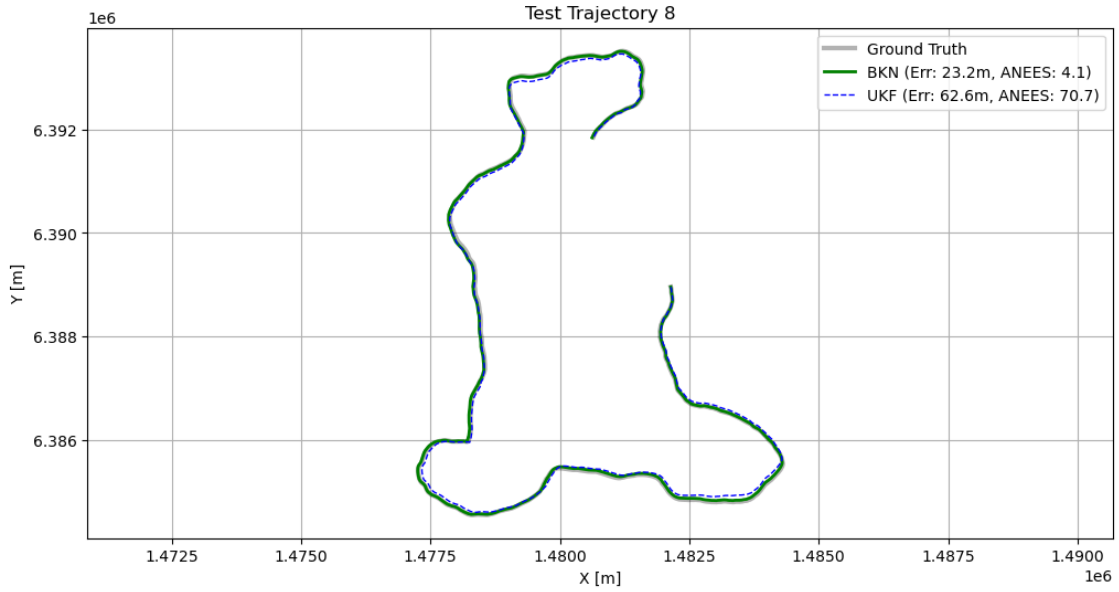
Evaluace: 25%| | 5/20 [00:40<02:00, 8.00s/it]



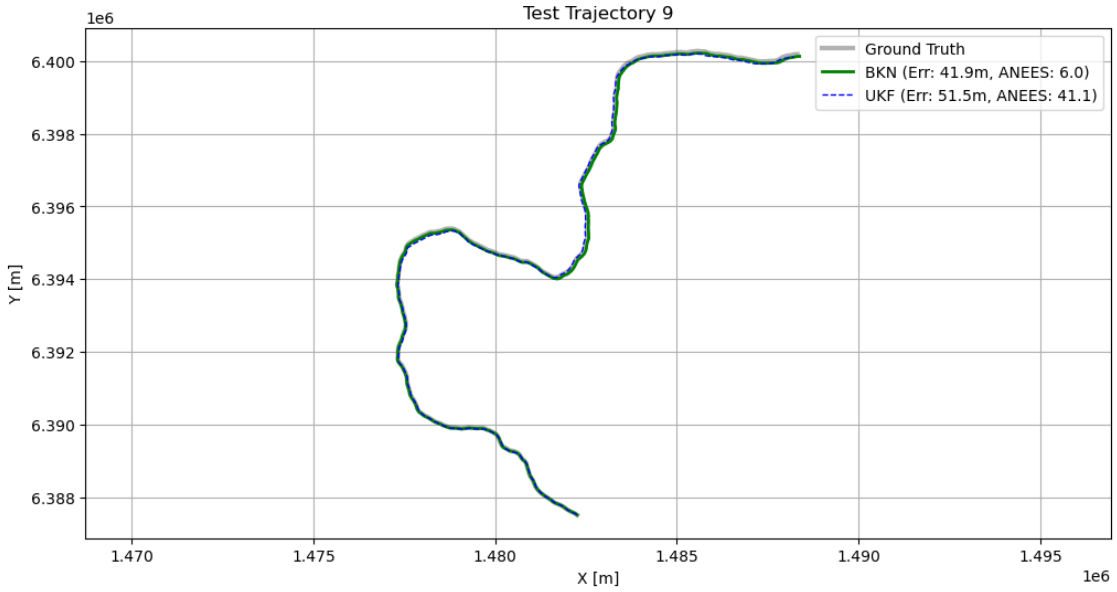
Evaluace: 30%| | 6/20 [00:48<01:52, 8.03s/it]



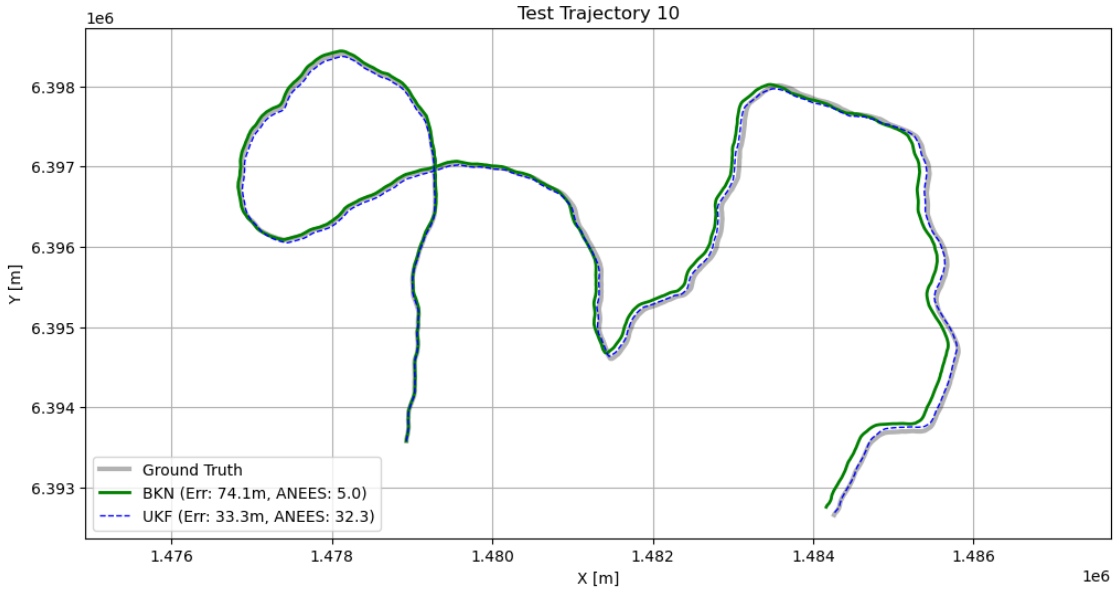
Evaluace: 35%| | 7/20 [00:56<01:44, 8.03s/it]



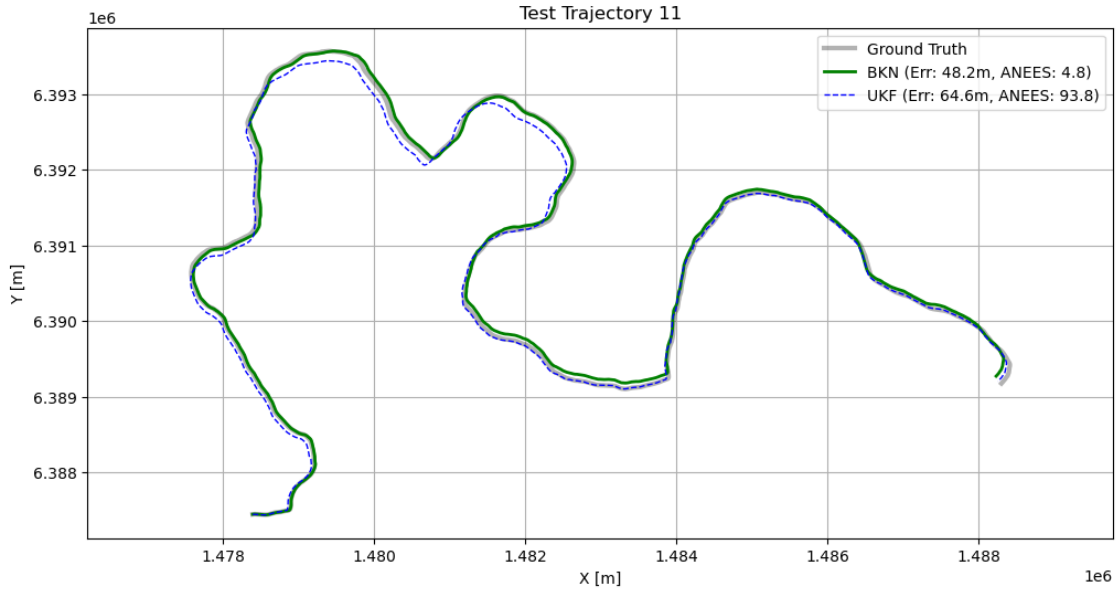
Evaluace: 40%| | 8/20 [01:04<01:35, 7.96s/it]



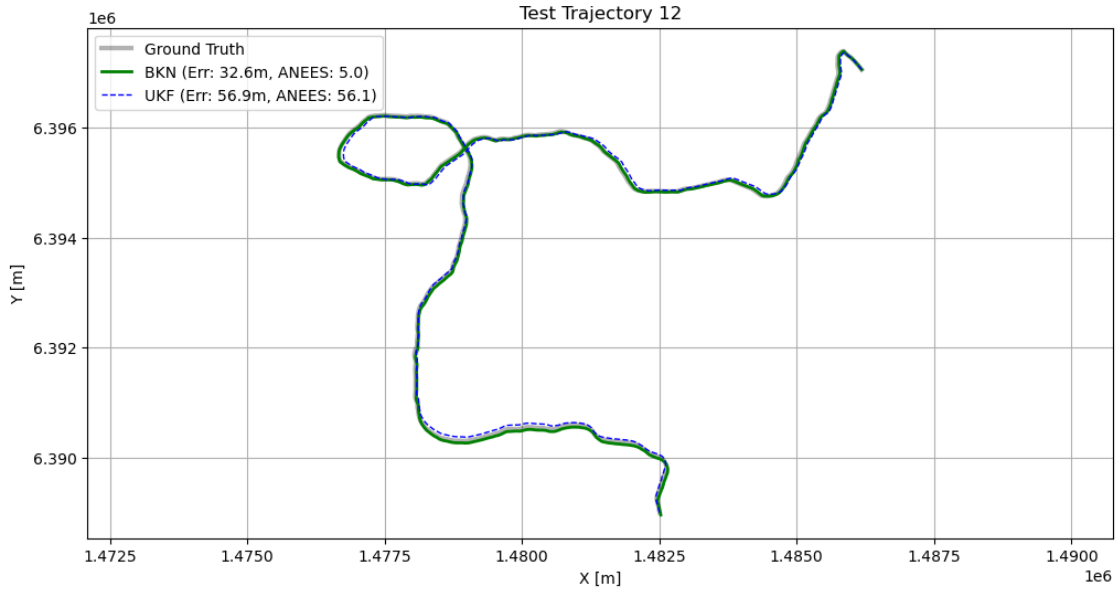
Evaluace: 45%| | 9/20 [01:12<01:28, 8.02s/it]



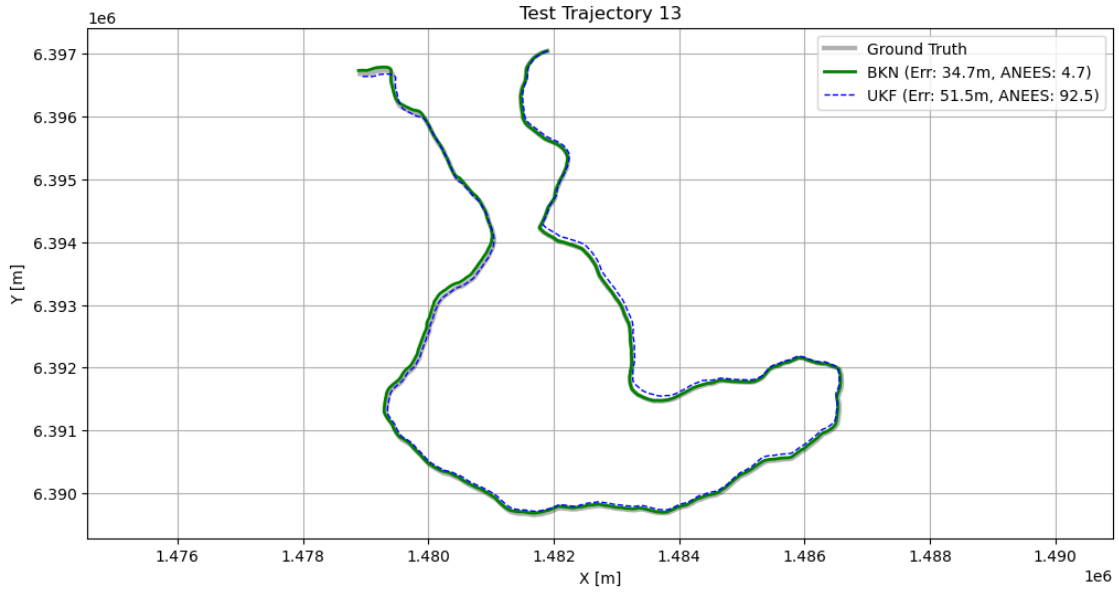
Evaluace: 50%| | 10/20 [01:20<01:20, 8.00s/it]



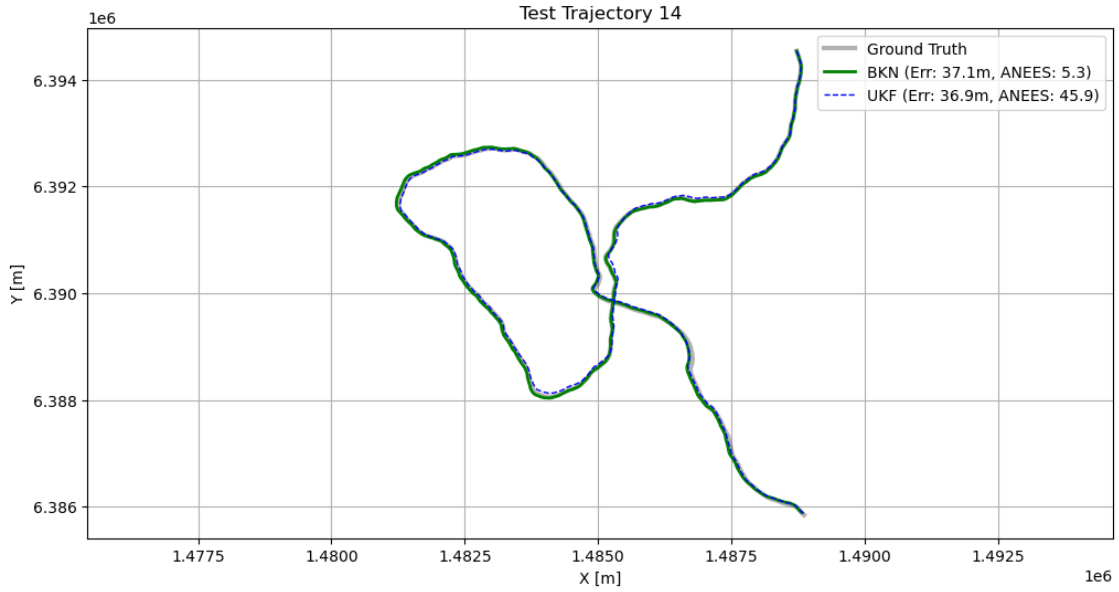
Evaluace: 55%| | 11/20 [01:28<01:12, 8.05s/it]



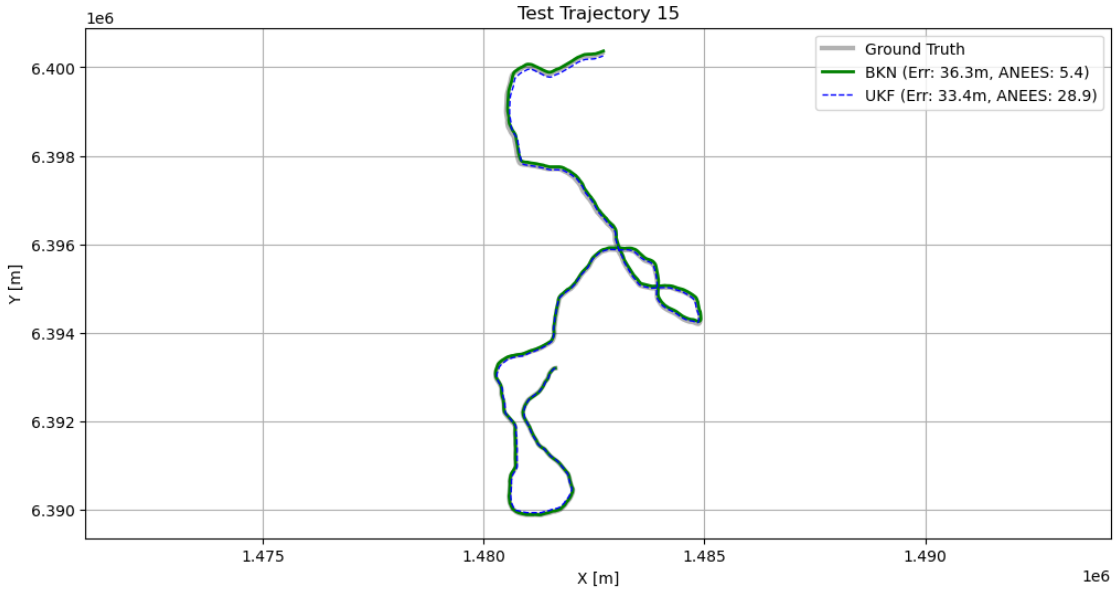
Evaluace: 60%| | 12/20 [01:36<01:03, 7.97s/it]



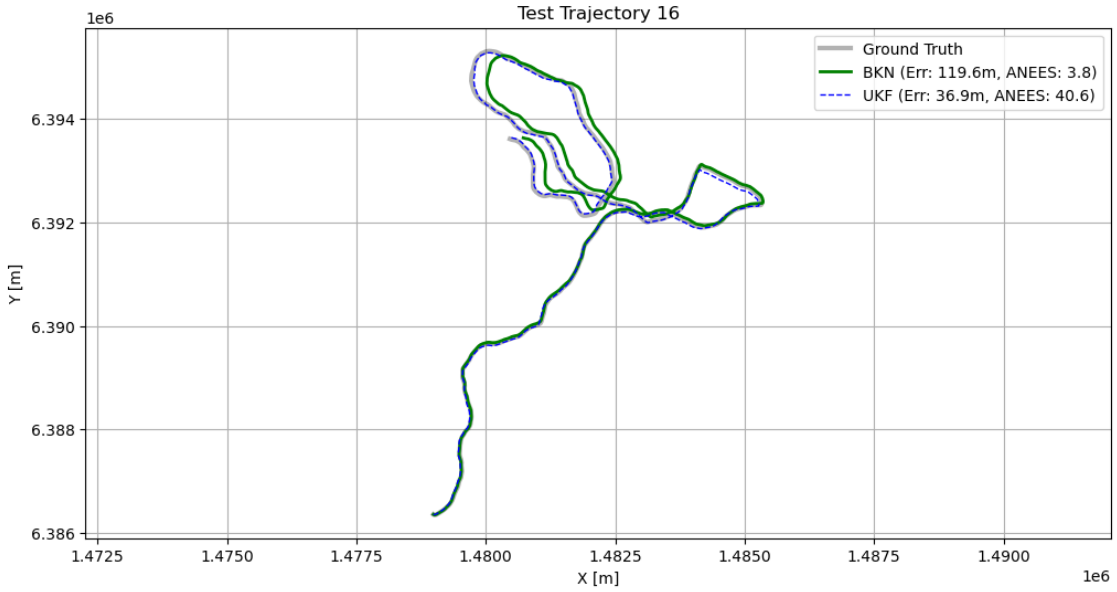
Evaluace: 65%| | 13/20 [01:44<00:55, 7.97s/it]



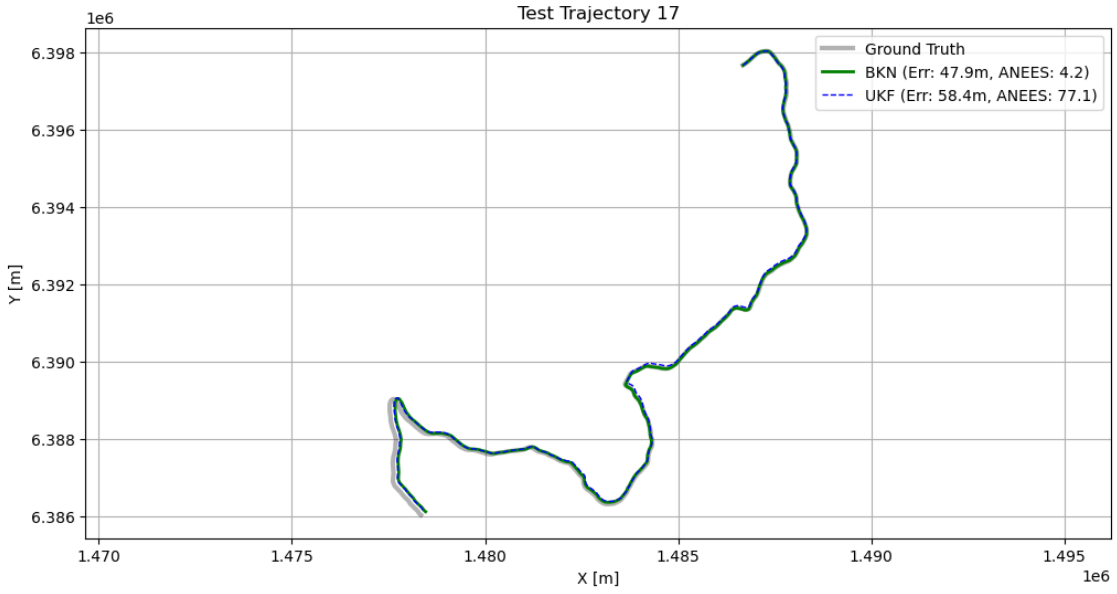
Evaluace: 70%| | 14/20 [01:52<00:48, 8.09s/it]



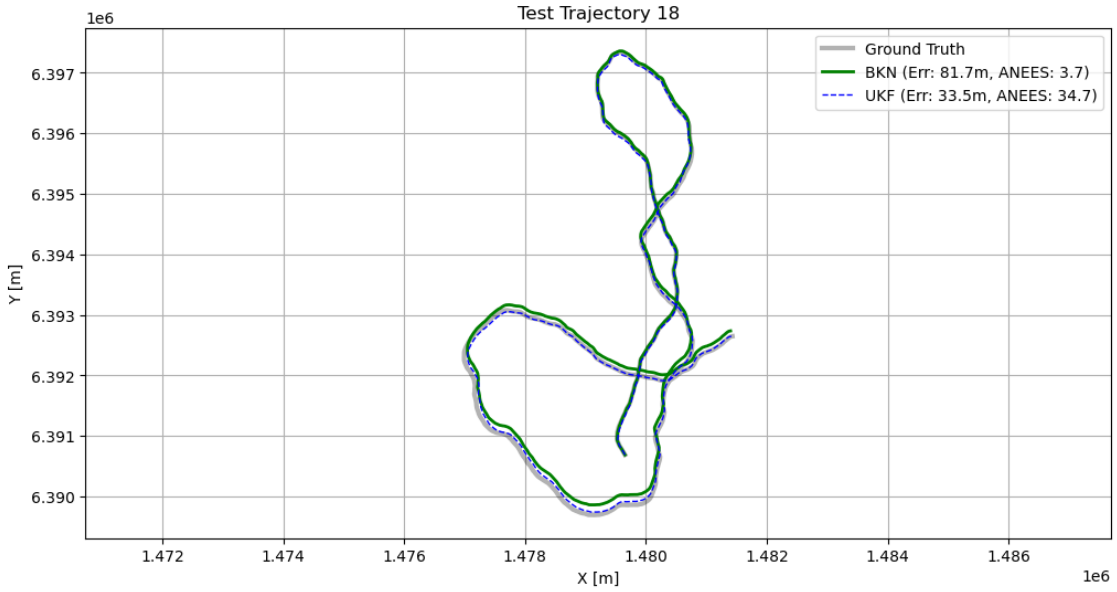
Evaluace: 75%| | 15/20 [02:00<00:40, 8.00s/it]



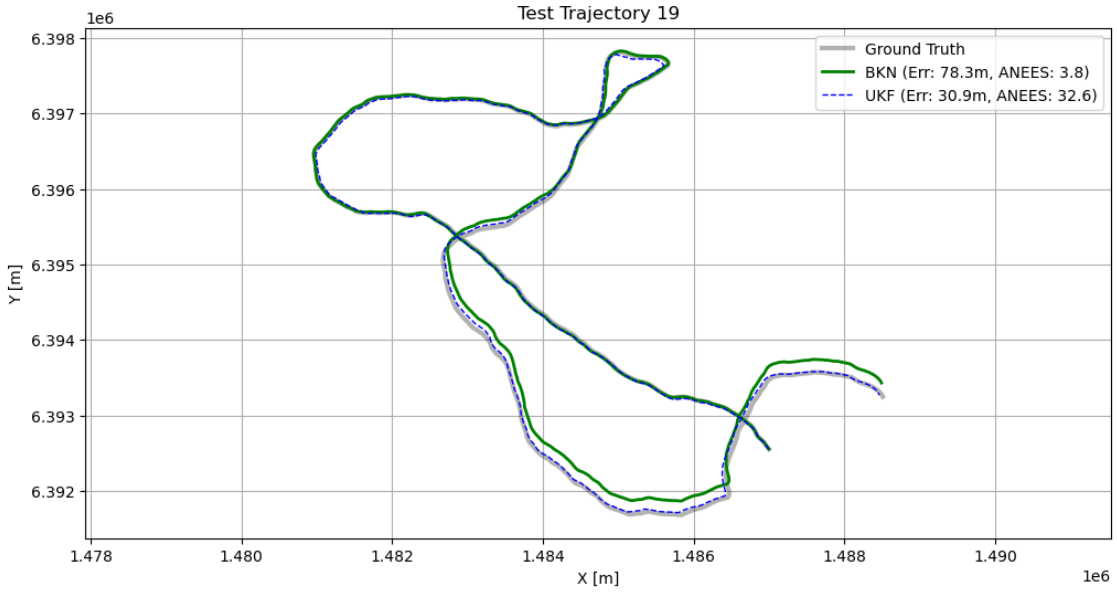
Evaluace: 80%| | 16/20 [02:07<00:31, 7.85s/it]



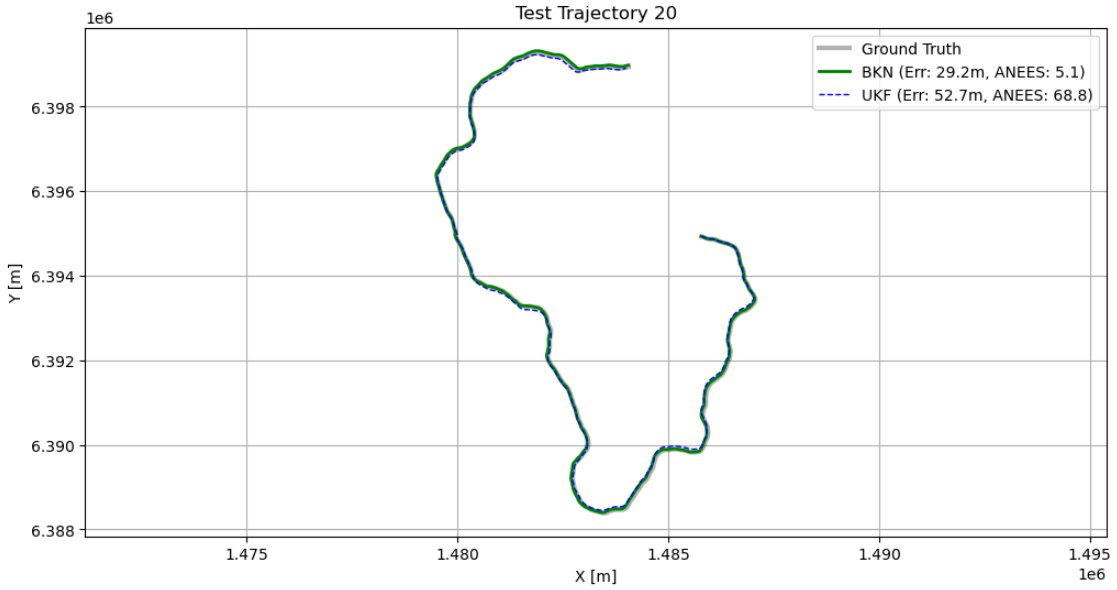
Evaluace: 85%| | 17/20 [02:15<00:23, 7.89s/it]



Evaluace: 90%| | 18/20 [02:23<00:16, 8.00s/it]



Evaluace: 95%| | 19/20 [02:31<00:07, 8.00s/it]



Evaluace: 100%| | 20/20 [02:39<00:00, 7.99s/it]

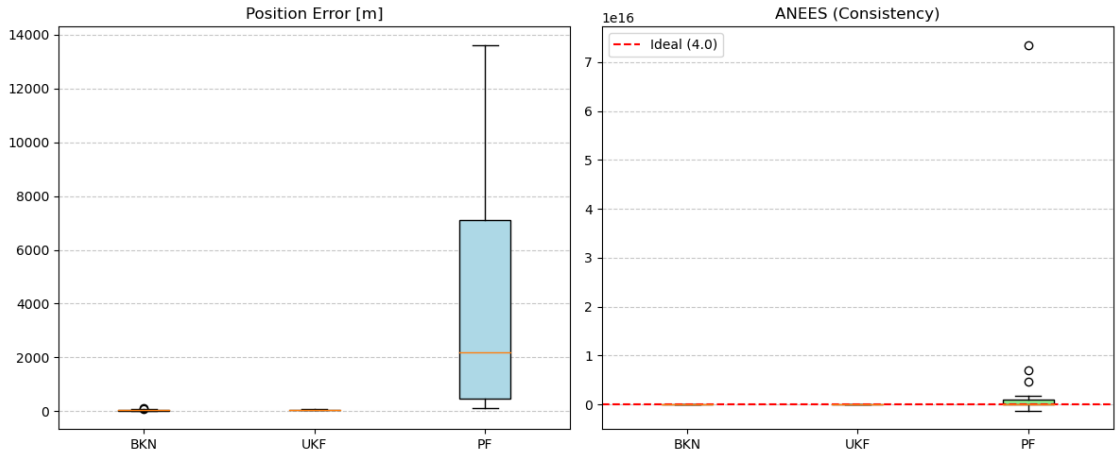
```
================================================================================
========================================
DETAILNÍ VÝSLEDKY (Pozice v metrech | ANEES - ideál ~4.0)
================================================================================
```

```
========================================
```

	Run_ID	BKN_PosErr	BKN_ANEES	UKF_PosErr	UKF_ANEES	PF_PosErr	APF_PosErr
0	1	14.89	4.43	38.17	33.33	3,974.27	41.25
1	2	34.59	5.38	43.13	54.50	378.18	33.95
2	3	23.76	4.80	42.63	35.61	726.13	31.72
3	4	23.55	4.50	55.35	35.62	7,860.12	42.68
4	5	20.06	3.70	41.38	41.92	8,221.14	70.22
5	6	28.84	5.05	34.53	27.40	134.73	46.55
6	7	37.51	4.77	45.29	71.28	178.18	101.68
7	8	23.21	4.15	62.59	70.68	12,373.03	93.30
8	9	41.91	5.96	51.53	41.13	495.99	114.56
9	10	74.06	4.95	33.32	32.27	641.41	83.99
10	11	48.23	4.83	64.65	93.80	869.59	126.77
11	12	32.55	4.99	56.92	56.08	13,083.67	172.61
12	13	34.70	4.69	51.49	92.51	6,852.95	121.93
13	14	37.13	5.32	36.88	45.86	245.90	69.02
14	15	36.34	5.40	33.41	28.93	3,492.97	99.10
15	16	119.61	3.79	36.90	40.57	13,615.81	94.75
16	17	47.88	4.18	58.40	77.15	709.49	439.95
17	18	81.73	3.66	33.45	34.65	370.42	37.97
18	19	78.34	3.84	30.94	32.58	3,981.16	105.53
19	20	29.16	5.12	52.75	68.82	4,190.43	38.99

```
================================================================================
========================================
SOUHRNNÁ STATISTIKA (20 trajektorií)
```

```
================================================================================
========================================
Model      | Pos Error [m] (Mean ± Std)     | ANEES (Mean ± Std)
--------------------------------------------------------------------------------
BKN        | 43.40 ± 25.34 m                | 4.67 ± 0.62
UKF        | 45.19 ± 10.44 m                | 50.73 ± 20.61
PF         | 4119.78 ± 4532.02 m                | 4370083782027506.50 ±
15967109332985878.00
APF        | 98.33 ± 86.77 m                | 608278.94 ± 1514871.88
================================================================================
========================================
```

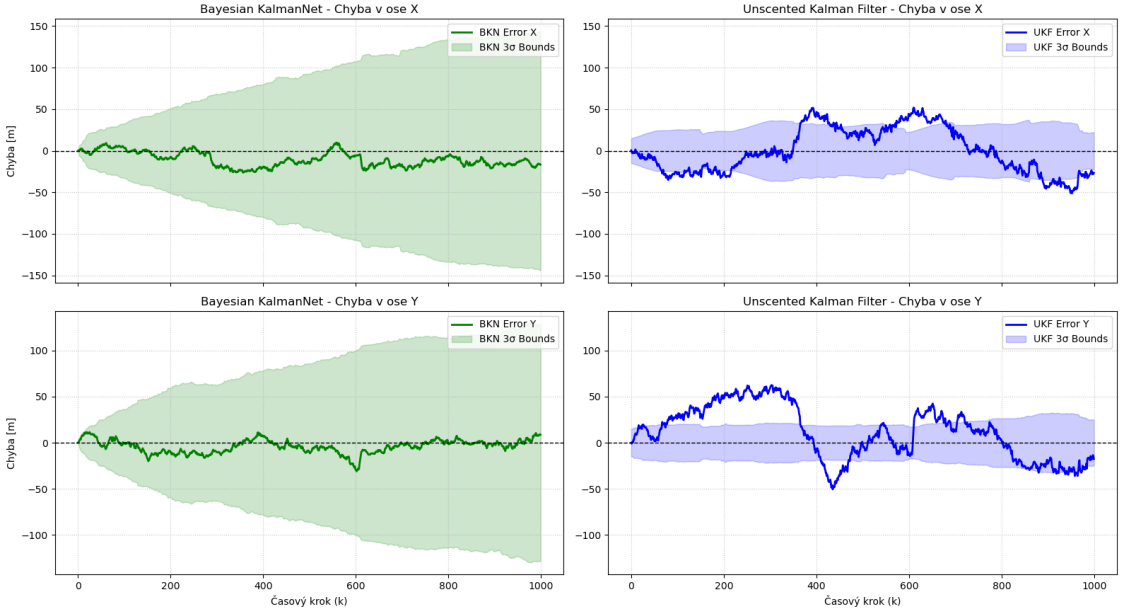


[22]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Zkontrolujeme, zda máme uložená data z předchozí buňky
if 'trajectory_history' not in globals() or len(trajectory_history) < 2:
    print(" Chyba: Chybí data 'trajectory_history'. Ujisti se, že jsi provedl␣
 ↪úpravu v hlavní smyčce.")
else:
    for traj_idx in range(2):
        data = trajectory_history[traj_idx]

        gt = data["gt"]
        bkn_est = data["bkn_est"]
        bkn_P = data["bkn_P"]
        ukf_est = data["ukf_est"]
        ukf_P = data["ukf_P"]

        seq_len = min(len(gt), len(bkn_est), len(ukf_est))
        time_steps = np.arange(seq_len)
```

```
        # Výpočet odchylek (Error = Odhad - Skutečnost)
        err_bkn_x = bkn_est[:seq_len, 0] - gt[:seq_len, 0]
        err_bkn_y = bkn_est[:seq_len, 1] - gt[:seq_len, 1]

        err_ukf_x = ukf_est[:seq_len, 0] - gt[:seq_len, 0]
        err_ukf_y = ukf_est[:seq_len, 1] - gt[:seq_len, 1]

        # Výpočet 3-sigma mezí z kovariančních matic
        # (index 0,0 je variance X; index 1,1 je variance Y)
        std_bkn_x = np.sqrt(bkn_P[:seq_len, 0, 0])
        std_bkn_y = np.sqrt(bkn_P[:seq_len, 1, 1])

        std_ukf_x = np.sqrt(ukf_P[:seq_len, 0, 0])
        std_ukf_y = np.sqrt(ukf_P[:seq_len, 1, 1])

        # --- VYKRESLOVÁNÍ ---
        fig, axs = plt.subplots(2, 2, figsize=(16, 10), sharex=True)
        fig.suptitle(f'Trajektorie {traj_idx + 1}: Analýza Konzistence (Chyba␣
↪vs. 3  Obálka)', fontsize=16)

        # 1. BKN - Poloha X
        axs[0, 0].plot(time_steps, err_bkn_x, 'g-', linewidth=2, label='BKN␣
↪Error X')
        axs[0, 0].fill_between(time_steps, -3*std_bkn_x, 3*std_bkn_x,␣
↪color='green', alpha=0.2, label='BKN 3  Bounds')
        axs[0, 0].axhline(0, color='black', linestyle='--', linewidth=1)
        axs[0, 0].set_title('Bayesian KalmanNet - Chyba v ose X')
        axs[0, 0].set_ylabel('Chyba [m]')
        axs[0, 0].legend(loc='upper right')
        axs[0, 0].grid(True, linestyle=':', alpha=0.7)

        # 2. BKN - Poloha Y
        axs[1, 0].plot(time_steps, err_bkn_y, 'g-', linewidth=2, label='BKN␣
↪Error Y')
        axs[1, 0].fill_between(time_steps, -3*std_bkn_y, 3*std_bkn_y,␣
↪color='green', alpha=0.2, label='BKN 3  Bounds')
        axs[1, 0].axhline(0, color='black', linestyle='--', linewidth=1)
        axs[1, 0].set_title('Bayesian KalmanNet - Chyba v ose Y')
        axs[1, 0].set_xlabel('Časový krok (k)')
        axs[1, 0].set_ylabel('Chyba [m]')
        axs[1, 0].legend(loc='upper right')
        axs[1, 0].grid(True, linestyle=':', alpha=0.7)

        # 3. UKF - Poloha X
        axs[0, 1].plot(time_steps, err_ukf_x, 'b-', linewidth=2, label='UKF␣
↪Error X')
```

```
        axs[0, 1].fill_between(time_steps, -3*std_ukf_x, 3*std_ukf_x,␣
↪color='blue', alpha=0.2, label='UKF 3 Bounds')
        axs[0, 1].axhline(0, color='black', linestyle='--', linewidth=1)
        axs[0, 1].set_title('Unscented Kalman Filter - Chyba v ose X')
        axs[0, 1].legend(loc='upper right')
        axs[0, 1].grid(True, linestyle=':', alpha=0.7)

        # 4. UKF - Poloha Y
        axs[1, 1].plot(time_steps, err_ukf_y, 'b-', linewidth=2, label='UKF␣
↪Error Y')
        axs[1, 1].fill_between(time_steps, -3*std_ukf_y, 3*std_ukf_y,␣
↪color='blue', alpha=0.2, label='UKF 3 Bounds')
        axs[1, 1].axhline(0, color='black', linestyle='--', linewidth=1)
        axs[1, 1].set_title('Unscented Kalman Filter - Chyba v ose Y')
        axs[1, 1].set_xlabel('Časový krok (k)')
        axs[1, 1].legend(loc='upper right')
        axs[1, 1].grid(True, linestyle=':', alpha=0.7)

        # Sladění Y-os pro spravedlivé porovnání mezi BKN a UKF
        max_y_val_x = max(np.max(np.abs(err_bkn_x)), np.max(3*std_bkn_x), np.
↪max(np.abs(err_ukf_x)), np.max(3*std_ukf_x)) * 1.1
        max_y_val_y = max(np.max(np.abs(err_bkn_y)), np.max(3*std_bkn_y), np.
↪max(np.abs(err_ukf_y)), np.max(3*std_ukf_y)) * 1.1

        axs[0, 0].set_ylim(-max_y_val_x, max_y_val_x)
        axs[0, 1].set_ylim(-max_y_val_x, max_y_val_x)
        axs[1, 0].set_ylim(-max_y_val_y, max_y_val_y)
        axs[1, 1].set_ylim(-max_y_val_y, max_y_val_y)

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        plt.show()
```

Trajektorie 1: Analýza Konzistence (Chyba vs. 3σ Obálka)



Trajektorie 2: Analýza Konzistence (Chyba vs. 3σ Obálka)