Here’s a high-level, practical overview of the codebase, how data flows end-to-end

**The overview of codes**

* **Real\_train.py-training loop**

Builds the dataset/loader, samples diffusion noise (EDM-style scalings), embeds inputs, runs the denoiser, computes a masked reconstruction loss on future steps, logs to TensorBoard, and triggers validation/plotting every N epochs.

* **infer\_2.py-validation & plotting pass**

Implements a 50-step denoising sampler with a Heun-like (predict-correct) update, fixed noise schedule (σ\_max=20 → σ\_min=0.002), and returns avg loss + one qualitative plot per run.

* **networks\_2.py-denoiser architecture**

Temporal per-agent processing (depthwise temporal conv + self-attention), light inter-agent attention (per time step), roadgraph encoder + cross-attention, then a head to predict [x,y,θ][x,y,\theta][x,y,θ]. Supports an **optional internal 128-d command embedding** (left/straight/right/null).

* **map\_pre\_old.py-data preprocessing & dataset**

Parses map/agent XML into tensors: trajectories [B,A,T,3][B,A,T,3][B,A,T,3] with masks, plus a roadgraph tensor [B,M,64][B,M,64][B,M,64] (xy, local tangents, curvature proxy…). Includes ego-centric transforms and fixed polyline resampling.

* **utiles.py-helpers**

• embed\_features: sinusoidal encodings for (x, y, θ), scenario time t, and diffusion time τ; returns [B, A, T, 5\*D] embeddings.  
• sample\_noise: adds noise to future (t≥10) and returns σ per sample.  
• plot\_trajectories: side-by-side predicted vs ground truth figure with map context.

* **metrics\_map.py –metrics & realism analysis**

Implements dataset-level metrics and grouped histograms (fixed ranges) such as linear/angular speeds, off-road/collision indicators, TTC (with “>90s” bin), and computes **pairwise Wasserstein distances** + permutation p-values (Holm–Bonferroni supported).

* **pca-trajectories.py-PCA & Sinkhorn score**

Flattens trajectories, does PCA, and measures distributional distance between real vs generated in PCA space via an entropy-regularized Sinkhorn distance (lower is better).

* **model\_output\_xml.py**

Runs the sampler (50 steps, σmin⁡=0.002\sigma\_{\min}=0.002σmin​=0.002), optionally smooths trajectories (Savitzky–Golay), transforms back to global frame, and **writes per-scene XML** for external comparison tools. Includes ADE/FDE helpers for exported sets.

**dataflow**

1. Load & preprocess  
   MapDataset (from map\_pre\_old.py) reads scenes, builds:
   * feature\_tensor [B,A,T,3] (x,y,θ), feature\_mask [B,A,T], and roadgraph tensors [B,M,64] (+ masks). Centering aligns scenes; θ is derived from velocity if missing.
2. Training (diffusion objective)
   * Noise the future part (t≥10) with sample\_noise, draw per-sample σ.
   * Compute EDM scalings c\_in, c\_out, c\_skip.
   * Embed inputs with embed\_features(inputs, σ) → [B,A,T,C\_in].
   * Forward: model(embedded, roadgraph\_tensor, feature\_mask, roadgraph\_mask) → denoised prediction for future.
   * Reconstruct x₀ via c\_out \* model\_out + c\_skip \* noisy\_future and compute MSE on valid timesteps (masked). Logs LR, grad-norm, losses, and runs periodic validation.
3. Inference / validation
   * Fixed 50-step σ schedule from 20.0 → 0.002.
   * Heun-style predict + correct using two forwards per step (except last), with the same EDM scalings at each σ.
   * After the final step, one more forward at σ\_final to estimate final\_predicted\_x0; compute masked MSE against GT, and produce a qualitative plot (map + pred + GT + initial noise).
4. Metrics & outputs (optional)
   * Use metrics\_map.py for realism stats & histograms; model\_output\_xmls.py to export XMLs of predicted trajectories (with optional smoothing).