Supplementary material

We provide in this Appendix some additional details to understand our work:

- We provide more details on our experimental setting in Sec. 7.
- We discuss how the Argoverse 2 Trust but Verify relates to our problem in Sec. 8.
- We provide the detailed precision tables and qualitative examples for our main results in Sec. 9.
- We study how the model behaves with exact map inputs in Sec. 10.
- We give pseudocode overviews of our two original MapEX modules in Sec. 11.
- We give a figure of a query based onlline HDMap estimation framework without MapEX modules in Fig. 4.

7. Detailed setting and codebase

We introduce here the detailed experimental details used for our experiments along with in-depth explaination of how existing maps are obtained for our various scenarios. Our code is largely based on the official MapTRv2 code[1], and will be made available along with our standalone MapModEX library on the MultiTrans project's official github: https://github.com/anr-multitrans.

Training details We largely reprise the 24 epochs training settings from our MapTRv2 [33] base, which were described in the original paper as:

> "ResNet50 is used as the image back- bone network unless otherwise specified. The optimizer is AdamW with weight decay 0.01. The batch size is 32 (containing 6 view images) and all models are trained with 8 NVIDIA GeForce RTX 3090 GPUs. Default training schedule is 24 epochs and the initial learning rate is set to 6×10-4 with cosine decay. We extract ground-truth map elements in the perception range of ego-vehicle following [...] The resolution of source nuScenes images is 1600×900. [...] Color jitter is used by default in both nuScenes dataset and Argoverse2 dataset. The default number of instance queries, point queries and decoder layers is 50, 20 and 6, respectively. For PV-to-BEV transformation, we set the size of each BEV grid to 0.3m and utilize efficient BEVPoolv2 [77] operation. Following [16], $\lambda_c = 2$, $\lambda_p = 5$, $\lambda_d = 0.005$. For dense prediction loss, we set $\alpha_d, \alpha_p, \alpha_b$ to 3, 2 and 1 respectively. For the overall loss, $\beta_o = 1$, $\beta_m = 1$, $\beta_d = 1$."

Our own training setting solely differs from MapTRv2's in the fact that we train on 2 NVIDIA Quadro RTX 8000 GPUs. This in turn mean we need to reduce the batch size by 4 and scale learning rates by 2 following standard scaling heuristics for Adam optimizers [15].

Scenario 1 implementation We remove the divider and pedestrian crossings from available HDMaps.

Scenario 2a implementation For each map element localization, we add noise from a Gaussian distribution with standard deviation of 1 meter. This has the effect of applying a uniform translation to each map element (dividers, boundaries, crosswalks).

Scenario 2b implementation For each ground truth point - keeping in mind a map element is made up of 20 such points - we sample noise from a Gaussian distribution with standard deviation of 5 meters and add it to the point coordinates.

Scenario 3a implementation We delete 50% of the pedestrian crossings and lane dividers in the map, add a few pedestrian crossings (half the amount of the remaining crossings) and finally apply a small warping distortion to the map. The warping distortion is composed of first trigonometric warping with horizontal and vertical amplitudes 1, and inclination 3. We then perform triangular warping following a slightly perturbed grid where each point on the regular grid is shifted according to random Gaussian noise with standard deviation 1.

Scenario 3b implementation For each map, we draw a uniform random value between 0 and 1. If it is below p=0.5 we keep the true HDMap, otherwise we perturb it in the same way as in Scenario 3a.

8. On the Trust but Verify dataset

The Argoverse 2 Trust but Verify (TbV) dataset [27] offers situations where the HDMap does not fit sensor inputs for change detection. Unfortunately, it is **not suitable** for our purposes it only says whether the current map fits sensor data (yes or no) **without giving the new map** (see Sec. 3.3 of [27] or the associated code). Without the relevant ground truth we cannot evaluate on it.

[1] https://github.com/hustvl/MapTR/tree/maptrv2

Additionally, while TbV is an excellent dataset for change detection, it unfortunately contains a limited number of real scenarios to train model for online HDMap acquisition. Moreover, a number of the change scenarios are indiscernible for our HDMap representation (e.g. change in the type of divider). Interestingly, the limited number of hand curated change situations is reserved for the validation and test sets with the train set generated from synthetic data. Where TbV chooses to generate synthetic views that differ from the available HDMap, we take the opposite view of modifying the HDMaps. While this is likely less desirable for change detection, it is of no consequence for online HDMap acquisition and much lighter computationally.

9. Fine grained results of map estimations

Tab. 6 provides a deeper look into the detailed results of MapEX and sheds light on how the different types of existing maps actually benefit the model.

Interestingly, the noisy **Scenarios 2a** and **2b** seem to help the model give a rough approximation of map elements (good scores for retrieval thresholds of 1.5m) but are less useful when it comes to predict very precise element localizations. As such, these scenarios appear to help the model by providing a **general idea of what the situation looks like**. Nevertheless, **Scenarios 2a** appears to still subtantially improve the base MapTRv2 model for precise element localizations at 0.5m (which is much lower than the standard deviation of the added noise).

Conversely, when outdated map **Scenarios 3a** and **3b** are useful to predict map elements, they tend to provide fairly precise element localizations (the gap between precision at 0.5m and 1.5m is lower). While these scenarios strongly improve performance at all precision thresholds, the improvement is also much larger for very precise element localizations. As such, they seem to work by providing a **more precise approximations of map elements**.

Scenario 1 (with only boundaries) for its part shines by providing near perfect estimations of map boundaries at all levels: it properly **identifies the provided road boundary localizations as perfectly accurate** and restitutes them as is. Interestingly, it also provides significant gains in precision at all retrieval thresholds for lane dividers and pedestrian crossings even though the existing map has no information on them.

10. Map change detection

We discuss here an additional module initially explored for MapEX. We include this discussion here as this module does not improve performance (and is therefore not an improved version of MapEX), but sheds light on what happens when perfect existing maps are available to the model.

10.1. Map change detector

There are a number of situations where fully accurate HDMaps might be mixed in with the imperfect HDMaps (e.g. our **Scenario 3b**). As such, we propose a lightweight change detection module to leverage these situations.

We introduce a learned change detection query token and perform cross-attention between this token and intermediate map element queries at different stages of the decoder. This token is then decoded by dense layer into a change prediction $c \in [0, 1]$ (with a sigmoid activation). At training time, we train this token with a binary cross entropy loss (with target $\hat{c} = 1$ if the map is not fully accurate and $\hat{c} = 0$ if it is): we minimize

$$\mathcal{L} = \mathcal{L}_{Base} + \mathcal{L}_{BCE}(c, \hat{c}), \tag{3}$$

with $\mathcal{L}_{Base}$ the loss of the base online HDMap estimator. At test time, if no change is detected we output the existing HDMap instead of the prediction (and we output the decoder predictions as usual if a change is detected).

Using the existing HDMap has two benefits: it provides a very precise HDMap (something most methods struggle with [11]), and it provides a way to stop the map estimation process early. Indeed, returning the existing map removes the need for further decoding of the query tokens which can be expensive.

10.2. Processing accurate existing maps

We take a closer look at how MapEX deals with perfectly accurate existing maps as it can sometimes happen in scenarios like **Scenario 3b**. To this end, we compare MapEX to variants that use an explicit map change detection module (described in Appendix 10) and substitute the predicted map with the input existing map if no change is detected. Tab. 7 shows MapEX does not need a change detection module: it recognizes and uses accurate existing map elements on its own. In fact, training a change detection module jointly with MapEX appears to deteriorate performance.

Table 7. Usefulness of the change detection module (**Scenario 3b**). MapEX seems to recognize and leverage existing maps without the need for explicit change detection.

Method	Average Precision at {0.5m, 1.0m, 1.5m}			
	$AP_{divider}$	AP_{ped}	$AP_{boundary}$	mAP
MapEX	92.8 ± 0.1	87.2 ± 0.1	99.3 ± 0.2	93.1 ± 0.1
... w/ substitution	92.5 ± 0.3	87.3 ± 0.3	99.4 ± 0.1	93.0 ± 0.1
... w/ sub. & optimization	92.5 ± 0.2	87.2 ± 0.2	99.3 ± 0.1	93.0 ± 0.1

11. Pseudo code

We provide here pseudo code for our two additional modules: the EX query encoding module (Alg. 1) and the pre-attribution code (Alg. 2).

Table 6. Detailed table of retrieval Precisions at different thresholds for the main results. Reproduced scores for the base MapTRv2 model are given for reference.

(a)

Method	$AP_{divider}$		
	$Precision^{0.5}_{divider}$	$Precision^{1.0}_{divider}$	$Precision^{1.5}_{divider}$
MapTRv2	46.0	66.4	75.4
MapEX-S1	50.7 ± 0.3	69.6 ± 0.7	77.8 ± 0.6
MapEX-S2a	*62.8 ± 2.1*	*83.6 ± 1.4*	92.1 ± 1.3
MapEX-S2b	50.9 ± 3.0	77.5 ± 2.2	89.0 ± 1.0
MapEX-S3a	76.2 ± 0.5	86.6 ± 0.3	*90.8 ± 0.3*
MapEX-S3b	**88.4 ± 0.5**	**93.8 ± 0.4**	**95.8 ± 0.2**

(b)

Method	AP_{ped}		
	$Precision^{0.5}_{ped}$	$Precision^{1.0}_{ped}$	$Precision^{1.5}_{ped}$
MapTRv2	34.5	65.1	78.8
MapEX-S1	38.8 ± 0.5	68.8 ± 0.5	80.0 ± 0.5
MapEX-S2a	*46.5* ± 0.5	85.5 ± 1.9	**97.2 ± 0.4**
MapEX-S2b	28.4 ± 0.5	72.3 ± 2.1	*91.7 ± 1.8*
MapEX-S3a	56.2 ± 0.4	*79.4 ± 0.5*	86.7 ± 0.7
MapEX-S3b	**77.7 ± 0.5**	**89.9 ± 0.3**	93.6 ± 0.3

(c)

Method	$AP_{boundary}$		
	$Precision^{0.5}_{boundary}$	$Precision^{1.0}_{boundary}$	$Precision^{1.5}_{boundary}$
MapTRv2	39.6	70.3	80.6
MapEX-S1	**99.8 ± 0.1**	**99.8 ± 0.1**	**100.0 ± 0.1**
MapEX-S2a	80.5 ± 0.9	96.4 ± 0.4	98.9 ± 0.2
MapEX-S2b	34.9 ± 0.2	75.8 ± 0.2	90.0 ± 0.3
MapEX-S3a	97.4 ± 0.3	**99.9 ± 0.1**	**100.0 ± 0.1**
MapEX-S3b	97.8 ± 0.4	**99.7 ± 0.3**	**100.0 ± 0.1**

Data: Map element $m^{EX} = \{(x_0^{EX}, y_0^{EX}), \ldots, (x_{L-1}^{EX}, y_{L-1}^{EX})\}$ of class c (among divider, crossing and boundary).
Result: list query_list of L H-dimensional EX queries.
query_list = [];
for $i \leftarrow 0$ **to** *L-1* **do**
  /* Encode position */
  pos_vec = array($[x_i^{EX}, y_i^{EX}]$);
  /* Encode class */
  class_vec = one_hot(c, num_class=3);
  /* Build query */
  pad_vec = zeros($H - 5$);
  query_i = concatenate([pos_vec, class_vec, pad_vec]);
  query_list.append(query_i);
end
return query_list;

Algorithm 1: Encoding map elements into EX Queries.

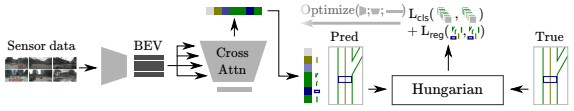

Figure 4. **Overview of a classic query based framework**. Sensor data is encoded into BEV features, before being cross attended with learned detection queries in a DETR-like scheme. The final attended queries serve to predict coordinates and classes of map elements. The model is trained using a Hungarian matching between predictions and ground truths.

Data: Predictions $p = \{p_i\}_{i=0,\ldots,49}$, (Padded) ground truths $g = \{g_i\}_{i=0,\ldots,49}$, correspondence list $c = \{c_i\}_{i=0,\ldots,49}$ where $c_i = -1$ if there is no correspondence
Result: Assignment $a = \{a_i\}_{i=0,\ldots,49}$ where a_i is the index of the ground truth associated to the i-th prediction.
/* Split off pre-attributed pairs */
$p^p, g^p, c^p, p^n, g^n, c^n, split\_inds$ = Split(p,g,c);
/* Perform Hungarian matching */
a^n = Hungarian(p^n, g^n);
/* Merge c^p with a^n */
a = Merge(p^p, a^n, split_inds); **return** a;

Algorithm 2: Hungarian matching with pre-attribution.