Learning Optimal Transport Plans Via Autoregressive Token Regression

Ivan Jaen Marquez **Sotirios Panagiotis Chytas** **Vikas Singh**
University of Wisconsin–Madison
ivanjaenm@cs.wisc.edu chytas@wisc.edu vsingh@biostat.wisc.edu
https://github.com/vsingh-group/TOR-OT

Abstract

Optimal Transport (OT) provides a principled framework for finding the minimum cost coupling between probability distributions. Classical solvers remain the gold standard for this task. At the same time, OT is an ideal testbed for studying whether learning-based amortized models can internalize the structure of OT type problems. Here, ground-truth solutions are available and marginal constraints are explicitly defined. Also, exact discrete solutions have well-characterized sparsity. We describe ToR-OT, a transformer-based model that learns to predict discrete OT plans through autoregressive token regression. We reframe OT into a structured sequence generation task. ToR-OT decodes sparse transport plans as sequences instead of solving an optimization task at test time. We focus on three properties: (a) Zero-shot generalization: pretraining on a large synthetic corpus of OT instances allows a single model to generalize to unseen distribution pairs and heterogeneous cost metrics without retraining, (b) Sparse primal decoding: by exploiting the inherent sparsity in exact discrete OT solutions, the decoder predicts only nonzero entries (avoids dense coupling outputs), (c) Amortized inference: on low-dimensional discrete problems, ToR-OT gives accurate approximate plans and compares favorably with per-instance neural OT baselines. Overall, ToR-OT is not yet a replacement for classical solvers, but a step toward understanding how autoregressive transformers can learn structure of constrained optimization problems like OT.

1 Introduction

Optimal Transport (OT) gives a principled framework for comparing probability distributions. It computes the minimum cost of transforming one measure into another, and finds use in fluid dynamics Chen et al. [2024], image processing Rabin and Papadakis [2015], brain connectivity analysis Ma et al. [2022], and machine learning, especially in domain adaptation and generative modeling Arjovsky et al. [2017]. Typically, OT solvers Friesecke [2024], Cuturi [2013] treat each instance independently, solving each pair of distributions from scratch. This instance-specific setup leaves open the question of whether some regularities shared across related problems can be learned and reused.

The above observation of learning/amortization has indeed been studied in the last few years, and a number of results have reported success with "neural" or learning-based approaches to OT Korotin et al. [2023], Li et al. [2024]. Neural Optimal Transport (NeuralOT) methods parameterize the dual potentials or transport maps using neural networks. In a number of situations, this allows solution of large-scale problems that can approach classical solvers. However, these approaches frequently remain instance-specific in a different sense. Each model is optimized to solve OT between *one specific pair of input measures* (e.g., a Gaussian source distribution and a mixture-of-Gaussians target distribution). It requires retraining or at minimum, fine-tuning when shown new source-target pairs. Meta OT Amos et al. [2023] mitigates these issues where a single model learns to predict dual potentials across multiple OT instances. This is accomplished by training on a distribution of problem

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

pairs (e.g., transporting between different pairs of MNIST digits). But Meta OT is still trained for specific problem domains: a model trained on image-to-image transport does not directly port over to probability distributions over labels. Other recent ideas Jiang et al. [2025] have explored embedding distributions into latent spaces to compute transport maps, offering further improvements. Despite these developments, zero/few-shot generalization across heterogeneous OT problem instances, e.g., different distribution or cost types, in the way that large language models generalize across diverse downstream tasks is not possible at this time to our knowledge.

On the other hand, there is broad agreement that transformer-based models show strong capabilities beyond the language domain. Recent works show that autoregressive transformers can solve arithmetic problems Lee et al. [2024], discover symbolic structures in partial differential equations Bhatnagar et al. [2025] and generate PDE solver code Li et al. [2025]. Surprisingly, a result last year showed that these models can even perform numerical regression by representing real values as token sequences Song and Bahri [2025]. This capability, called decoding-based regression, enables transformers trained with standard cross-entropy loss to predict "numeric" outcomes with good accuracy, comparable to traditional regression methods. This suggests that autoregressive pretraining could transfer to structured prediction problems whose outputs admit a meaningful sequential representation.

The above discussion raises a natural question: can autoregressive token regression, when shown a sufficiently large sample size of examples during pretraining, learn to solve optimization problems beyond simple regression? We study this question in the context of OT, a structured prediction task with well-defined ground truth solutions, a clear optimization objective, and a nice geometric structure. Unlike standard regression where outputs are single scalars, OT requires predicting sparse transport plans (matrices with constrained marginals). So, it is a compelling test case for whether transformer models can provide sensible results.

Contributions. We give evidence that autoregressive token prediction, the mechanism behind LLMs, can be successfully adapted to solve OT-type structured optimization problems when shown sufficient training data. Our ToR-OT is not positioned as a replacement for classical OT solvers. Instead, we use OT as a controlled setting to investigate the degree to which transformers can internalize structural regularities in such constrained optimization problems. OT is useful for this purpose because its solutions are globally coupled, ground-truth optima can be obtained by classical solvers, and its polyhedral structure gives well-characterized sparsity. To this end, our experiments test distinct scientific hypotheses about learned OT structure, including causal interventions/linear probes that test whether the model represents and uses marginal-capacity information during generation. We present a model that treats transport plan prediction as sequence generation rather than numerical optimization. We achieve (a) zero-shot generalization across unseen distribution pairs and (b) amortized efficiency through row-wise autoregressive decoding that predicts only nonzero entries.

2 Background

We first review two concepts used throughout the paper: (i) the discrete OT solution of interest (that we aim to predict), and (ii) a token regression module for predicting numerical outputs.

2.1 Optimal Transport

Let $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{m} b_j \delta_{y_j}$ be two empirical measures, where $a \in \Delta_n := \{a \in \mathbb{R}_+^n : \sum_i a_i = 1\}$ and $b \in \Delta_m := \{b \in \mathbb{R}_+^m : \sum_j b_j = 1\}$ are probability mass vectors supported on points $\{x_i\}_{i=1}^n \subset \mathcal{X}$ and $\{y_j\}_{j=1}^m \subset \mathcal{Y}$ respectively. Let $C \in \mathbb{R}_+^{n \times m}$ denote a ground-cost matrix with entries $C_{ij} = c(x_i, y_j)$ (e.g., Euclidean distance $C_{ij} = \|x_i - y_j\|_2$). In the discrete OT setting, the output is a *transport plan* $\pi \in \mathbb{R}_+^{n \times m}$ where π_{ij} gives the amount of mass moved from x_i to y_j.

Based on this definition, we can distinguish two variations: the (i) **Monge** and (ii) **Kantorovich** formulations. The Monge formulation (Equation (1)) seeks a deterministic assignment $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\}$ and enforces *no mass splitting*. The Kantorovich relaxation (Equation (2)) replaces the map σ with a matrix π that can split mass across targets and satisfies linear marginal constraints. The feasible set $\Pi(a, b) = \{\pi \geq 0 : \pi \mathbf{1}_m = a, \pi^\top \mathbf{1}_n = b\}$ is a convex polyhedron in $\mathbb{R}^{n \times m}$.

$$\min_{\sigma} \left\{ \sum_{i=1}^{n} a_i C_{i,\sigma(i)} \;\middle|\; b_j = \sum_{i:\,\sigma(i)=j} a_i \;\; \forall j \right\}. \quad (1) \qquad \min_{\pi \geq 0} \left\{ \langle \pi, C \rangle \;\middle|\; \pi \mathbf{1}_m = a,\; \pi^\top \mathbf{1}_n = b \right\}. \quad (2)$$

Between the two formulations above, we write our prediction task around the Kantorovich *coupling matrix* π instead of the Monge *map* σ. While a valid Monge map (1) may not exist when $n \neq m$ or when mass weights are unequal, the Kantorovich relaxation (2) guarantees a feasible solution for any (a, b) pair. Also, its optimal coupling matrix satisfies a fundamental sparsity property that makes sequence prediction a good fit:

Property 2.1 (Sparsity)**.** *For the discrete Kantorovich problem* (2)*, an optimal plan π^* exists with at most $n + m - 1$ nonzeros [Peyré and Cuturi, 2019, Prop. 3.4].*

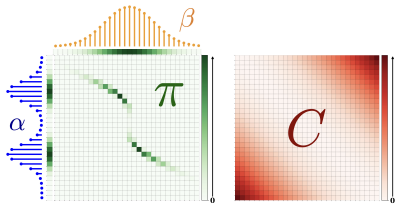

Figure 1: Toy discrete OT instance showing the source measure a, target measure b, cost C, and an optimal plan $\pi^\star \in \mathbb{R}_+^{n \times m}$

We now have the main object of interest: the prediction target ($\pi \in \mathbb{R}_+^{n \times m}$), shown in Figure 1.

2.2 Token Regression (Decoding-Based Regression)

As briefly mentioned above, decoding-based regression Song and Bahri [2025] represents a real-valued target $y \in \mathbb{R}$ using a finite sequence of discrete tokens $(t_1, \ldots, t_K)$ that encodes y under a chosen numeric format, for example, a string such as `10.452` or `1e-10`. A causal decoder Jurafsky and Martin [2023] then models the conditional distribution over the token sequence given an input representation $\phi(\mathbf{x}) \in \mathbb{R}^d$ via the standard autoregressive factorization:

$$p_\theta(t_1, \ldots, t_K \mid \phi(\mathbf{x})) = \prod_{k=1}^{K} p_\theta(t_k \mid \phi(\mathbf{x}), t_{<k}), \quad (3)$$

where $t_{<k} = (t_1, \ldots, t_{k-1})$ denotes the prefix tokens. Training follows the standard language model machinery of optimizing with next-token cross-entropy. Notice that by representing numbers as short token sequences and training with next-token prediction, decoding gives numeric strings parsed to $\mathbb{R}$.

Remark 2.1 (Variable-length outputs). Although originally set up for scalar regression, the same mechanism can easily output *variable-length* structured objects, e.g., lists, tuples, or sparse representations. We will use exactly this idea to serialize sparse OT plans.

3 Why Is OT a Structured Prediction Problem?

We can start by asking: why would predicting an OT plan be difficult for standard neural heads? One possible reason is that the transport plan π which solves (2) is a globally *constrained* object. Every entry must satisfy nonnegativity ($\pi_{ij} \geq 0$), row marginals ($\sum_j \pi_{ij} = a_i$) for all i, as well as column marginals ($\sum_i \pi_{ij} = b_j$) for all j. These constraints couple all the entries because changing π_{ij} requires compensating adjustments elsewhere. Otherwise, we will lose feasibility. Further, the sparsity pattern varies across instances because some solutions concentrate mass on few edges and others "spread" it.

Possible mismatch? We know that a dense regression head which outputs all $n \times m$ entries assumes (i) fixed output dimension (but n, m vary across instances) and (ii) independent predictions (but entries are constrained). Further, sparsity is ignored although typical plans have $\mathcal{O}(n + m)$ nonzeros, not $\mathcal{O}(nm)$. We reframe OT plan prediction as a *structured output* with instance dependent complexity. While this is ill-suited for fixed size dense regressors, it is natural for variable-length sequences.

Although Property 2.1 above is usually viewed as a geometric property of the transportation polytope, for sequence modeling one benefit is that any extreme-point OT plan can be described in an encoding of length $\mathcal{O}(n+m)$ rather than $\mathcal{O}(nm)$. Basically, instead of a dense $n \times m$ matrix, we can represent the plan as a sparse list of $(\text{source}, \text{target}, \text{mass})$ triples:

$$\pi \;\equiv\; \{(i_k, j_k, w_k)\}_{k=1}^{K}, \qquad K \leq n + m - 1, \quad (4)$$

where each triple means moving mass w_k from source i_k to target j_k.

Serializing description for autoregressive decoding. We will simply predict the plan *row by row*: for each source i, emit (target index, mass) pairs until the row budget a_i is allocated, then proceed to row $i+1$. Autoregression will condition on previous rows, implicitly tracking remaining column capacities (the b_j not yet consumed). Formally, for row i, we can emit:

$$\mathrm{Row}_i = \{(j_\ell^{(i)}, w_\ell^{(i)})\}_{\ell=1}^{K_i}, \quad \text{where} \quad \sum_{\ell=1}^{K_i} w_\ell^{(i)} = a_i. \tag{5}$$

Now, conditioning on $\mathrm{Row}_{<i}$ shows to the decoder the remaining column-capacity state, allowing it to learn column-marginal behavior without an explicit constraint enforcement. The main benefit is that sparsity allows a compact serialization: essentially, OT plans become short sequences (length $\mathcal{O}(n+m)$) aligned with autoregressive generation.

3.1 Reformulating OT as a Seq2Seq Task

We cast OT as a sequence prediction task, an approach we call **ToR-OT** (**To**ken **R**egression for **OT**). Given a problem triplet (μ, ν, c) which includes source distribution, target distribution, and cost type, the model produces a token sequence encoding the sparse plan π via the row-wise representation (5).

Autoregressive decoding fits this structure for two reasons. First, the output length is inherently variable: different instances induce different numbers of active entries, so a decoder emitting tokens until an `EOS` marker will avoid the inefficiency of fixed-size outputs. Second, the row-major serialization allows a clean probabilistic factorization. Generating row i while conditioning on rows $\pi_{<i,:}$ allows implicit tracking of remaining target capacities. Formally, the decoder models

$$p(y \mid \mu, \nu, c) \;=\; \prod_{t=1}^{T} p(y_t \mid y_{<t}, \mu, \nu, c), \tag{6}$$

where $y = (y_1, \ldots, y_T)$ is the complete token sequence encoding all rows.

Remark 3.1. We do *not* solve an optimization problem at test time. We will *generate* a structured sequence whose semantics encode an OT plan.

3.2 Will ToR-OT Generalize? Test-Time Compute Cost?

The sequence formulation yields two main benefits at inference time, which we discuss below.

1. **Test-time compute efficiency.** Rather than running a new solve for each query, ToR-OT amortizes inference into generating a short token sequence, with no iterative optimization. Because only nonzero entries are emitted, we can avoid $\mathcal{O}(nm)$ dense outputs and the sequence length scales as $\mathcal{O}(n+m)$.

2. **Zero-shot generalization.** Second, because the source and target measures (μ, ν) are passed directly as input context, a single pretrained model can generalize to new distribution families without test-time retraining.

Remark 3.2 (Amortization and zero-shot generalization). ToR-OT is trained on *solved* OT instances: tuples (μ, ν, c, π^*) where μ, ν are sampled uniformly at random and π^* is the optimal plan from a classical solver. At test time, the model is evaluated on *new* distribution family pairs *unseen* during training (e.g., Laplace source, Gamma target), and must generalize the transport structure learned across training pairs to predict plans for these unseen combinations. This is different from solving each instance from scratch or training separate networks per fixed (μ, ν) pair. ToR-OT amortizes learning *across* distribution families, enabling zero-shot inference on unseen pairs.

4 ToR-OT: Tokenization, Set-to-Sequence Architecture, and Training

We now give details for ToR-OT: it is a transformer-based model that predicts the OT plan $\pi(\mu, \nu)$ given two input measures μ, ν and a cost metric c (e.g., ℓ_2, ℓ_2^2). The plan is generated one row at a time (Figure 2): for row i, a set encoder will embed the remaining problem (the unallocated source bins and residual target capacities), and a causal Transformer decoder will emit the nonzero entries of $\hat{\pi}_{i,:}$ as a short token sequence. The masses of the row are then subtracted from the target masses, and the next row is generated.

Serializing rows as token sequences. A row is simply the token sequence:

$$\texttt{BOS}\ \big(\langle t_j\rangle\ \langle E_k\rangle\ d_1\ d_2\ d_3\ d_4\big)^*\ \texttt{EOS},$$

with one group of six tokens per nonzero entry, in increasing target index j. The pointer token $\langle t_j\rangle$ identifies the target bin, and the mass is written in scientific notation, $\hat{\pi}_{ij} = d_1.d_2d_3d_4 \times 10^{-k}$: that is, an exponent token $\langle E_k\rangle$, $k \in \{1, \ldots, 9\}$, followed by four mantissa digits (masses below 10^{-9} are dropped). For example, a source bin i sending mass 0.0318 to target 11 and 0.0064 to target 12 is

$$\texttt{BOS}\ \langle t_{11}\rangle\ \langle E_2\rangle\ 3\ 1\ 8\ 0\ \langle t_{12}\rangle\ \langle E_3\rangle\ 6\ 4\ 0\ 0\ \texttt{EOS}.$$

All other entries in row i are implicitly zero.

Remark 4.1 (Design rationale). A small, fixed alphabet (e.g., $|\mathcal{V}| = 15$) ensures: (i) stable training across varying support sizes n, m, (ii) arbitrary numeric precision via digit strings, and (iii) compatibility with standard causal language modeling.

Remark 4.2 (Sequence length). A row with K_i nonzero entries will have $6K_i + 2$ tokens. For problems where optimal plans are sparse (Property 2.1), the outputted rows remain relatively short, typically $K_i = \mathcal{O}(1)$ entries per row, giving total sequence length $\mathcal{O}(n + m)$ tokens.

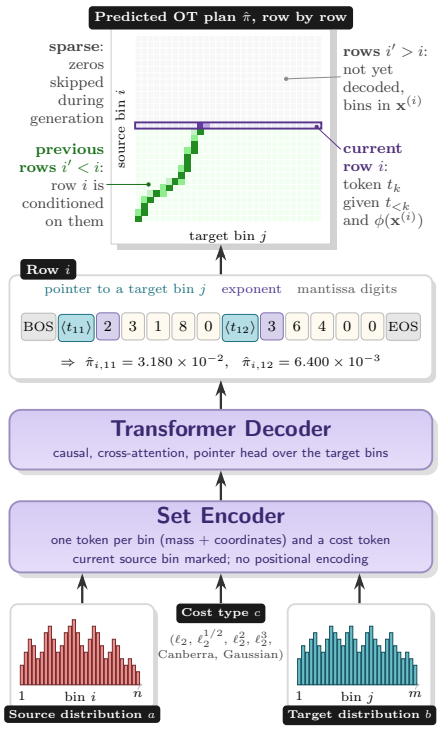

Figure 2: ToR-OT architecture. Motivated by Song and Bahri [2025], we view Optimal Transport as a sparse token-based multi-regression problem. Given a pair of source and target distributions, we train a Transformer to predict the nonzero values of the OT plan π^* as a sequence of digit tokens.

4.1 Encoding Residual Transport State $\phi(\mathbf{x}^{(i)})$

The key question is what information the decoder must condition on to generate feasible or near-feasible rows. For row i, the encoder receives the *remaining problem*

$$\mathbf{x}^{(i)} = \Big(c,\ \{(x_{i'}, a_{i'})\}_{i'=i}^{n},\ \{(y_j, r_j^{(i)})\}_{j=1}^{m}\Big), \qquad r_j^{(i)} = b_j - \sum_{i'<i} \hat{\pi}_{i'j}, \tag{7}$$

that is, the cost type c, the source bins not yet decoded with their masses, and all target bins with their remaining capacities. It maps $\mathbf{x}^{(i)}$ to one latent state per element: $\phi(\mathbf{x}^{(i)}) = \big(h_c,\ h_i^{\text{src}}, \ldots, h_n^{\text{src}},\ h_1^{\text{tgt}}, \ldots, h_m^{\text{tgt}}\big)$. The decoder attends to all of these states, with the target states h_j^{tgt} forming the candidate set for pointer selection (Section 4.2).

Tracking residual state across rows. Row i is conditioned on the past rows through the remaining capacities $r_j^{(i)}$ (7). Once the past rows are fixed, the completion of the plan depends on them through $r^{(i)}$. Since the encoder ignores input order, bin i carries a learned marker. We also retain the unallocated source bins $i' > i$ in the input so the model can account for downstream demand. Basically, if a later source bin can reach a given target at lower cost, the current row should preserve capacity for it.

Coordinate-based bin embeddings. Each bin becomes one token, embedded from its coordinate and its mass (Appendix A), and the encoder is a Transformer without positional encoding. A bin is thus identified by its coordinate (not by its place in the input), and so, relabeling the bins only permutes the states. The same model therefore works for any support size within its training range, and the same architecture also works for two-dimensional supports.

Conditioning on heterogeneous cost metrics. The cost type c enters as one additional learned token. Every bin attends to it, and a single model covers multiple costs. To summarize, $\phi(\mathbf{x}^{(i)})$ provides the geometry, the remaining capacities and the cost one needs to generate row i.

4.2 Autoregressive Decoding Under Marginal Constraints

The decoder needs to be set up to answer the question whether row-wise generation will/can ignore global constraints (column marginals). Our decoder conditions on *both* the encoder embedding $\phi(\mathbf{x})$ and previous row tokens. The prefix encodes used column capacities: if $\sum_{i'<i} \hat{\pi}_{i'j}$ mass has been allocated to target j, the model learns to respect the remaining capacity $r_j^{(i)} = b_j - \sum_{i'<i} \hat{\pi}_{i'j}$. We do *not* hard-enforce the constraints. In terms of architecture, the decoder is just a standard causal Transformer with L layers, H attention heads, and hidden dimension d. It receives $\phi(\mathbf{x})$ as a prefix and generates the token sequence $\mathbf{t} = (t_1, \ldots, t_K)$ via:

$$p_\theta(t_1, \ldots, t_K \mid \phi(\mathbf{x})) = \prod_{k=1}^{K} p_\theta(t_k \mid \phi(\mathbf{x}), t_{<k}). \tag{8}$$

For each row i, the decoder outputs (index, mass) pairs until an `EOS` token signals completion.

Remark 4.3 (Feasibility). Autoregressive generation of row i conditions on $\hat{\pi}_{<i,:}$ and the problem instance (μ, ν, c). The prefix $\pi_{<i,:}$ encodes used column capacities, guiding the model toward marginal-respecting plans. However, ToR-OT does not enforce constraints via projection. Feasibility is learned from training data, and so is not guaranteed. We will discuss this shortly. When strict feasibility is required, predicted plans can be post-processed.

To close up this section, we provide the details of setting training data and points not covered above.

4.3 Synthetic Pretraining Corpus Generation and Training

We now describe in more detail where the ground-truth token sequences come from. We generate a large synthetic corpus of OT instances as follows. First, we sample source and target distributions uniformly at random. Then, we sample the cost type c from $\{\ell_2, \ell_2^{1/2}, \ell_2^2, \ell_2^3, \text{Canberra}, \text{Gaussian}\}$. We then solve the Kantorovich LP with POT's network simplex solver and keep only the nonzero entries of the optimal plan π^*, i.e., `res = ot.solve(C, a, b)` and $\pi^* =$ `res.sparse_plan`.

Remark 4.4 (Sanity checks). Before tokenization, we verify that training targets satisfy: nonnegativity: $\pi^*_{ij} \geq 0$ for all i, j as well as row and column constraints. Plans failing these checks ($< 0.1\%$ of generated instances, due to numerical errors) are discarded.

Remark 4.5 (Coverage across random distributions). The training corpus spans diverse problem characteristics such as bin sizes, 1D/2D distributions, multiple cost metrics covering six different geometries. This diversity provides *transport structure*, i.e., how mass flows respond to cost and marginal changes rather than memorizing specific instance types.

Training. Our training objective is the standard cross-entropy loss over the generated token sequence. Given the digit-level tokenization, this transforms the task into a series of discrete classification tasks, $\mathcal{L}(\theta) = -\sum_{k=1}^{K} \log p_\theta(t_k \mid \phi(\mathbf{x}), t_{<k})$. To handle long sequences from row-wise serialization, during training our implementation uses FlashAttention-2 Dao [2024].

4.4 Decoding Complexity and Runtime Scaling

As described earlier, for Kantorovich formulation (2), there is always an optimal plan solution π with at most $n + m - 1$ nonzero entries Peyré [2025], and the number of predicted nonzero bins is $\mathcal{O}(n)$. So, we can distinguish a *worst-case* (highly uneven sparsity) and *average-case* (approximately uniform sparsity) runtime of a transformer-based model. Even when the *total* number of nonzeros is $\mathcal{O}(n + m)$, they can still be concentrated in a few rows. For the balanced case $n = m$, we can have $K_1 = \mathcal{O}(n)$ and $K_i = \mathcal{O}(1)$ for $i > 1$ while still satisfying $\sum_i K_i = \mathcal{O}(n)$. Then, $\sum_{i=1}^{n} K_i^2 = \mathcal{O}(n^2)$ so row-wise decoding with full self-attention has worst-case cost $\mathcal{O}(n^2)$. If each row has a bounded number of targets, $K_i = \mathcal{O}(1)$, then $\sum_i K_i = \mathcal{O}(n)$ and $\sum_{i=1}^{n} K_i^2 = \mathcal{O}(n)$. This means linear scaling in the self-attention under row-wise decoding. Here, the inference cost is dominated by the model's fixed per-token cost (and not the quadratic output size).

4.5 Context Relative to Classical and Dual Solvers

ToR-OT is a prototype for studying whether autoregressive models can internalize globally coupled constraints (and is not intended to replace exact solvers yet). We distinguish between two settings:

(a) When ToR-OT helps. ToR-OT is useful when solving many similar OT problems repeatedly, as in matching problems that recur within the same application domain, e.g., as in Pachauri et al. [2012]. In this regime, the formulation amortizes learning across instances and can produce accurate approximate plans without per-instance neural optimization. Because we decode row-wise, generating a complete plan requires sequential row predictions rather than a single dense matrix output.

(b) When ToR-OT is a sub-optimal fit. Classical OT solvers are always preferable when we seek certified optimality, or when constraints/costs fall far outside the training distribution. ToR-OT is also not a good fit if the plan is effectively dense (where sequence lengths grow and the advantage of predicting only nonzeros shrinks), such as in entropic OT Cuturi [2013], where adding an entropy regularizer spreads mass across many edges.

Primal versus dual OT prediction. In the learned OT setting, one key modeling choice is whether to learn OT structure in the primal or dual space. Several learning-based methods such as Meta OT and UNOT predict vector-valued *dual* potentials to avoid predicting the full matrix-valued coupling. These dual formulations typically use an ε-regularized OT objective: for a fixed $\varepsilon > 0$, one solves the regularized problem accurately, but it is an approximation of the original *non-entropic* Kantorovich problem. The choice of ε introduces a bias–stability tradeoff since larger ε values improve smoothness and numerical stability, while smaller values better approximate exact OT (albeit harder to optimize). ToR-OT uses the exact primal formulation of the Kantorovich LP (Equation (2)) and predicts its sparse primal coupling. Thus, while dual prediction avoids the dense $n \times m$ coupling by learning vector-valued potentials, ToR-OT instead uses the inherent sparsity of the exact primal solution to obtain a compact sequence representation of the coupling itself.

5 Experiments and Results

We evaluate ToR-OT across a range of settings to test what a transformer model learns when trained on OT data. The first four tasks test predictive and structural properties of the learned model and the last asks whether the model uses OT's constraint structure during generation. We ask whether ToR-OT: (1) generalizes to unseen distributions; (2) gives fast per-query inference through amortized pretraining, relative to per-instance neural baselines; (3) handles several cost types with a single model; (4) yields approximately feasible plans with row-wise autoregressive decoding; and (5) uses the remaining marginal capacities during generation rather than pattern-matching inputs to outputs.

5.1 Experimental Setting

Model. All one-dimensional results come from a single encoder–decoder Transformer (157.6M parameters; 12 heads, 6 encoder and 12 decoder layers) that generates the plan row by row. It is trained once, on 5M instances covering six cost types (ℓ_2, $\ell_2^{1/2}$, ℓ_2^2, ℓ_2^3, Canberra and Gaussian), support sizes from 5 to 100 bins and marginals drawn uniformly from the probability simplex, with the exact plans of POT as targets. There is no fine-tuning per cost, evaluation set or support size. Training took 24.4 wall-clock hours with data parallelism on 8 NVIDIA A100 GPUs. A second model with the same architecture and recipe is trained on two-dimensional histograms (Section 5.4). Additional training and evaluation details are in Appendix A, and ablations in Appendix E.

Baselines. We treat the exact solver of **POT** Flamary et al. [2021] as the ground truth and compare against three neural OT methods: **NeuralOT** Korotin et al. [2023], which trains a network for every new pair, and two amortized solvers of entropic OT, **Meta OT** Amos et al. [2023] and **UNOT** Geuter et al. [2025]. As both assume a fixed cost, we train Meta OT and UNOT separately for each cost with the official code, choosing the regularization $\varepsilon \in \{0.01, 0.001\}$ on a held-out validation set, and reporting the mean over three training seeds.

Evaluation. Problems have $n = m = 30$ bins ($n = 100$ and 200 in Appendix E.1). The **test set** has 1,000 random pairs per cost from the training distribution (unseen instances); the **OOD set** has the 144 ordered pairs of 12 named distributions, each pair binned on a common grid of 30 bins that covers both distributions, so the narrower one can have empty bins (Appendix F). We report the Mean Absolute Percentage Error (MAPE), $\sum_{ij} |\hat{\pi}_{ij} - \pi_{ij}| / \sum_{ij} \pi_{ij}$, of each method against the solution of the problem it solves. Meta OT and UNOT solve entropic OT and are compared with the converged Sinkhorn plan at their ε. ToR-OT and NeuralOT target the exact OT and are compared

with the optimal plan closest to their prediction, since the optimal plan is not always unique and POT returns one of them. Other metrics and every cost are in Appendix F.

5.2 One-Dimensional Distributions

We first evaluate two experimental families of one-dimensional distributions: 1. **Test set**, and 2. **OOD**. The test set corresponds to 1,000 held-out test pairs per cost of randomly generated source and target distributions, similar in nature to the training set used for the pretraining. OOD stands for 144 pairs of well-defined distributions (e.g., Beta, Cauchy, Exponential, Laplace) not seen during training.

5.2.1 Test-Set Samples

Figure 3 (top row) reports test-set performance. On the left, we see that ToR-OT consistently produces more accurate solutions than the baselines across all six cost functions, each method scored against the solution of the problem it solves.

Table 1: Reduction of ToR-OT's mean MAPE relative to Meta OT, the strongest baseline, for every cost ($n = 30$).

Cost	ℓ_2	$\ell_2^{1/2}$	ℓ_2^2	ℓ_2^3	Canberra	Gaussian
Test set	88%	91%	95%	93%	91%	64%
OOD set	60%	65%	75%	67%	69%	69%

Meta OT is the strongest baseline on every cost, and ToR-OT reduces its error by roughly an order of magnitude on five of the six costs (Table 1). The Gaussian cost is the hardest for ToR-OT, yet it still clearly improves on Meta OT there. UNOT and NeuralOT are far less accurate in this setting, even though NeuralOT is trained separately on every pair. All six costs are handled by a single ToR-OT model, without per-cost training.

5.2.2 Out-of-Distribution Samples

Out-of-Distribution (OOD) samples are pairs of well-defined distributions, such as Normal, Exponential, and Beta, that differ strongly from the randomly generated train-set distributions. Evaluating on OOD samples tests both applicability to common distributions and generalization to unseen instances, ruling out memorization. Indeed, OOD accuracy keeps improving with training and, in a smaller-scale study, is unchanged when the model is trained on only a tenth of the corpus (Appendix E). In the OOD setting (Figure 3, bottom row), ToR-OT's error grows relative to the test set on five of the six costs, but it remains the lowest of all methods on every cost, 60–75% below that of Meta OT (Table 1). Figure 4 (B) shows the relative MAPE improvement over Meta OT under ℓ_2^2 for 100 of the 144 distribution pairs; over all 144, ToR-OT is more accurate on 134, with a median improvement of 91%. Per-pair results for the other five costs, and every method (for all metrics), are in Appendix F.

5.2.3 Runtime

While classical solvers like POT are optimized for small-scale instances, ToR-OT operates in the amortized regime. In the runtime comparison we can ask a more specific question: after pretraining, how does a single amortized sequence model compare to per-instance neural OT methods? NeuralOT

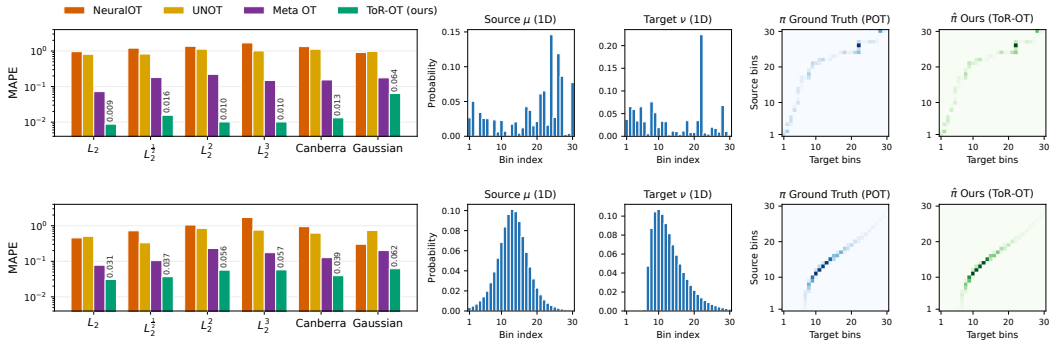

Figure 3: Test set (top row) results and OOD (bottom row) results. (left) ToR-OT vs NeuralOT, Meta OT and UNOT. On all six costs, our model performs favorably. (right) a sample of the evaluation set, along with its ground truth and predicted OT plan. In the case of the test set, both source and target distributions are randomly sampled, while in the OOD case, they correspond to well-defined distributions, like Beta and Exponential.

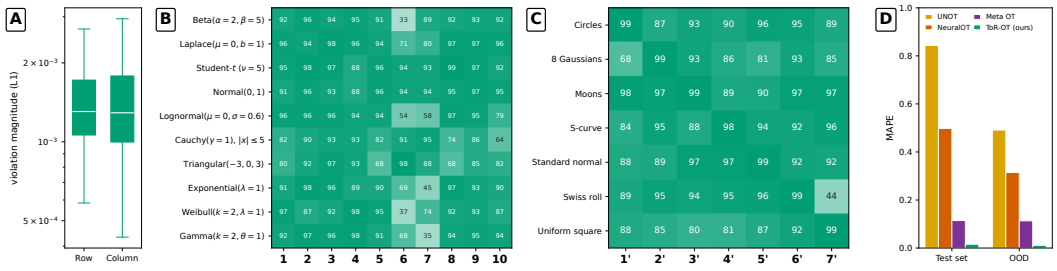

Figure 4: **(A)** Violation of marginal constraints in ToR-OT across rows and columns. **(B)** Relative improvement of ToR-OT's MAPE over Meta OT, the strongest baseline, under ℓ_2^2 for 100 of the 144 pairs of OOD one-dimensional distributions (all 144 are in Appendix F). The difference is always positive and reaches up to 99%. X-axis labels: 1: Beta, 2: Laplace, 3: Student's t, 4: Normal, 5: Lognormal, 6: Cauchy, 7: Triangular, 8: Exponential, 9: Weibull, 10: Gamma. **(C)** The same for all 49 pairs of OOD two-dimensional distributions. The difference is always positive and reaches up to 99%. X-axis labels: 1': Circles, 2': 8 Gaussians, 3': Moons, 4': S curve, 5': Standard Normal, 6': Swiss roll, 7': Uniform square. **(D)** Mean MAPE of every method on the 2D test and OOD sets.

requires a dedicated optimization loop for each new pair, whereas ToR-OT generates a plan in 1.09 s on one A100 GPU. Including its 24.4 h of training, ToR-OT is cheaper than NeuralOT after fewer than 100 instances. Meta OT and UNOT are faster still, but they solve entropic OT and their MAPE is at least 2.5 times ToR-OT's on every cost and set. The runtime result should therefore be viewed as some evidence for amortization relative to per-instance neural baselines (not a replacement for classical OT solvers). More runtime details can be found in Appendix B.

5.2.4 Feasibility of predicted plans

Despite no hard constraint enforcement, ToR-OT yields near-feasible plans. Figure 4 **(A)** reports the violation magnitudes on rows $\|\hat{\pi}\mathbf{1} - a\|_1$ and columns $\|\hat{\pi}^\top \mathbf{1} - b\|_1$ on the test set. The median for both is about 0.1% of the total mass, and no instance exceeds 1%. Column residuals, however, vary more across instances than row residuals, on every test and OOD set, and pooled over all sets, row residuals are also slightly smaller. Both effects are consistent with the autoregressive structure: each row is generated with its own mass as input, so its sum is controlled directly, while every column sum accumulates the errors of all rows. When improved feasibility is needed, two options are possible: a lightweight Sinkhorn post-processing step and a symmetric two-pass approach. Full details and ablations for both strategies are in Appendix C.

5.3 Mechanistic Interpretability

Our goal here is to determine whether the transformer represents and uses structural regularities of OT. Three experiments (Appendix D) give good evidence. We use a *causal intervention*, modifying the generation prefix to exhaust a column's budget. This forces $r_j^{(i)} = b_j - \sum_{i'<i} \hat{\pi}_{i'j} = 0$, while leaving model weights and the problem instance unchanged. We see that the model adapts routing in 99% of cases, tracking remaining capacity as a continuous quantity. *Linear probing* recovers remaining column capacities from final-layer hidden states with $R^2 = 0.7135$. Accuracy rises as generation progresses, ruling out static input encoding. *Attention pattern analysis* over 144 maps shows three consistent head types: global constraint heads, running row-budget accumulators, and spatial allocation heads. These results are consistent with the model representing and using OT constraint state during generation.

5.4 Extension to Multivariate Distributions: The Two-Dimensional Case

Finally, we extend the evaluation to the case of 2D histograms and focus on in-distribution test performance and OOD generalization. The OOD evaluation set comprises 49 pairs of well-defined 2D distributions (e.g. Standard Normal, S curve). This setting highlights that ToR-OT translates to 2D without any fundamental changes to the architecture (by flattening 2D bins into 1D). The only additional input is a 2D-specific cost for the model to capture the underlying geometry (Figure 5).

The 2D model shares the architecture and training recipe of the 1D model and is trained on 4M pairs of 2D histograms on $k \times k$ grids (k from 3 to 10). We evaluate on a 6×6 grid ($n = m = 36$), with a

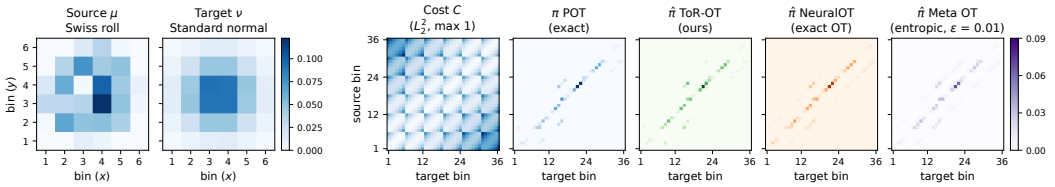

Figure 5: ToR-OT, NeuralOT and Meta OT on a 2D OOD pair (Swiss roll → Standard Normal under ℓ_2^2): the 6×6 histograms, the cost matrix and the plans over the 36 flattened bins, on one color scale. The exact same formulation as 1D is used. The only difference is that the 2D model is trained on 2D histograms, with each bin embedded from its 2D coordinate; the six costs are the same as in 1D.

test set of 1,000 random pairs per cost. Per pair, ToR-OT is more accurate than Meta OT on all 49 OOD pairs, with a median improvement of 93% and at least 44% (Figure 4 (C)). On the input pair of Figure 5, ToR-OT's plan follows the sparse structure of the exact plan, while NeuralOT's has a larger error. These gains are consistent across all six costs, where ToR-OT outperforms Meta OT on every OOD pair of distinct distributions (Appendix G).

6 Related work

The "neural" OT methods primarily focus on parametrizing the transport map with neural networks Seguy et al. [2018], Korotin et al. [2023]. They usually exploit a connection between dual OT formulation and NNs, derived from Brenier's theorem Brenier [1991]. Proposals in Bunne et al. [2022] and Uscidda and Cuturi [2023] leverage this idea to propose novel extensions such as inferring OT maps for multiple pairs of measures, some of them with a pre-specified context. One common feature this first set of neural OT methods share is that they seek to solve individual problem instances, similar to traditional optimization solvers. There is no learning from previously solved instances. This limitation has motivated a shift toward learned, amortized solvers that can transfer structure across instances. Approaches such as Meta OT Amos et al. [2023], Asadulaev et al. [2024] are good representatives: by training on a distribution of OT problems, they make inference on new pairs faster. They learn shared structure across tasks, and related approaches embed distributions into a latent space to produce problem-adapted OT maps or operators. While this improves reuse, it is often domain-based, since one usually restricts it to a fixed cost or fixed support size. A more recent result, Universal Neural OT Geuter et al. [2025] targets generalization across OT problems under a fixed cost. It uses Fourier Neural Operators (FNO) as a discretization-invariant architecture that can operate on inputs of varying resolution. To our knowledge, LLMs/Transformer models to solve OT problems in the style of ToR-OT have not been reported.

7 Conclusion and Limitations

We described ToR-OT, a novel transformer-based token-regression formulation for predicting OT plans between probability distributions. We have shown that ToR-OT can effectively internalize the structure of OT and provide highly accurate predictions on both the test set and out-of-distribution samples, while offering amortized runtime efficiency. One takeaway is that sparse *primal* OT plans can be effectively learned by sequence models. Operating in the exact discrete setting gives provably sparse plans (at most $n+m-1$ nonzeros, with no regularization bias), instance-dependent variable support that a standard autoregressive model handles naturally. And since ground truth is available, the mechanistic analysis reported here becomes possible. This is quite different from the neural OT literature's focus on continuous transport maps or entropy-regularized couplings.

Limitations. While we view ToR-OT as a promising step towards the study of transformers for OT, there are some limitations. First, scaling to larger supports (more bins) may benefit from richer source–target interactions in the encoder, e.g., attention biased by pairwise costs, which the model currently infers from bin coordinates and the cost type. Second, extending to higher-dimensional measures remains challenging: our study focuses on 1D/2D settings; a unified model that transfers across dimensions or operates on arbitrary (non-grid) supports would be a meaningful step toward more general learned OT solvers. Third, handling more general cost structures is required to better capture instance-specific geometry, especially under variable or data-dependent costs.

Acknowledgments and Disclosure of Funding

Marquez, Chytas and Singh were supported in part by NIH R01 AG092220.

References

Xinan Chen, Helene Benveniste, and Allen Tannenbaum. Unbalanced regularized optimal mass transport with applications to fluid flows in the brain. *Scientific Reports*, 14, 01 2024. doi: 10.1038/s41598-023-50874-y.

Julien Rabin and Nicolas Papadakis. Non-convex relaxation of optimal transport for color transfer between images. In Frank Nielsen and Frédéric Barbaresco, editors, *Geometric Science of Information*, pages 87–95, Cham, 2015. Springer International Publishing. ISBN 978-3-319-25040-3.

Kai Ma, Xuyun Wen, Qi Zhu, and Daoqiang Zhang. Optimal transport based ordinal pattern tree kernel for brain disease diagnosis. In Linwei Wang, Qi Dou, P. Thomas Fletcher, Stefanie Speidel, and Shuo Li, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*, pages 186–195, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-16437-8.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 214–223. JMLR.org, 2017.

Gero Friesecke. *Optimal Transport: A Comprehensive Introduction to Modeling, Analysis, Simulation, Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2024. doi: 10.1137/1.9781611978094. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611978094.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf.

Alexander Korotin, Daniil Selikhanovych, and Evgeny Burnaev. Neural optimal transport. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=d8CBRlWNkqH.

Zezeng Li, Shenghao Li, Lianbao Jin, Na Lei, and Zhongxuan Luo. Ot-net: a reusable neural optimal transport solver. *Mach. Learn.*, 113(3):1243–1268, January 2024. ISSN 0885-6125. doi: 10.1007/s10994-023-06493-9. URL https://doi.org/10.1007/s10994-023-06493-9.

Brandon Amos, Giulia Luise, Samuel Cohen, and Ievgen Redko. Meta optimal transport. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Mingchen Jiang, Peng Xu, Xichen Ye, Yun Yang, and Yifan Chen. Embedding empirical distributions for computing optimal transport maps. In *2025 IEEE International Symposium on Information Theory (ISIT)*, pages 1–6. IEEE, 2025.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dsUB4bst9S.

Rohan Bhatnagar, Ling Liang, Krish Patel, and Haizhao Yang. From equations to insights: Unraveling symbolic structures in pdes with llms, 2025. URL https://arxiv.org/abs/2503.09986.

Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. Codepde: An inference framework for llm-driven pde solver generation, 2025. URL https://arxiv.org/abs/2505.08783.

Xingyou Song and Dara Bahri. Decoding-based regression. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=avUQ8jguxg.

G. Peyré and M. Cuturi. *Computational Optimal Transport: With Applications to Data Science*. Foundations and trends in machine learning. Now Publishers, 2019. ISBN 9781680835502. URL https://books.google.com/books?id=J0BiwgEACAAJ.

Daniel Jurafsky and James H Martin. *Speech and Language Processing*. Pearson Higher Ed, 2023.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Gabriel Peyré. Optimal transport for machine learners, 2025. URL https://arxiv.org/abs/2505.06589.

Deepti Pachauri, Maxwell Collins, Risi Kondor, and Vikas Singh. Incorporating domain knowledge in matching problems via harmonic analysis. In *Proceedings of the... International Conference on Machine Learning. International Conference on Machine Learning*, volume 2012, page 1271, 2012.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *J. Mach. Learn. Res.*, 22(1), January 2021. ISSN 1532-4435.

Jonathan Geuter, Gregor Kornhardt, Ingimar Tomasson, and Vaios Laschos. Universal neural optimal transport. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=t10fde8tQ7.

Vivien Seguy, Bharath Bhushan Damodaran, Remi Flamary, Nicolas Courty, Antoine Rolet, and Mathieu Blondel. Large scale optimal transport and mapping estimation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1zlp1bRW.

Yann Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on Pure and Applied Mathematics*, 44(4):375–417, 1991. doi: https://doi.org/10.1002/cpa.3160440402. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160440402.

Charlotte Bunne, Andreas Krause, and Marco Cuturi. Supervised training of conditional monge maps. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Théo Uscidda and Marco Cuturi. The monge gap: A regularizer to learn all transport maps. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 34709–34733. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/uscidda23a.html.

Arip Asadulaev, Alexander Korotin, Vage Egiazarian, Petr Mokrov, and Evgeny Burnaev. Neural optimal transport with general cost functionals. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gIiz7tBtYZ.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=3X2EbBLNsk.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We support our claims on OT generalization, amortized efficiency through the six core hypothesis tested in Section 5 Experiments and Results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Section 7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This is an empirical paper and we do not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 detail the model, dataset and training routine. Appendix A describe the specific settings used for all experiments. The code for the experiments and plots will be released upon acceptance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We have disclosed all the information needed to reproduce the main experimental results of the paper. The code will be open-sourced upon acceptance, together with the instructions on how to reproduce the experiments.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: The technical details pertaining to the exact setup are given in Section 4 and appendix A. The code with instructions on how to run the full experiments will be open-sourced upon acceptance.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Box plots and error bars are given and described where relevant.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided sufficient information on the total computer resources needed for training our model in experimental part of Section 6. Similarly we report inference runtimes.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have conducted research under the NeurIPS Code Of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our research aims to understand weather transformers can internalize the OT structure. It is a first step toward amortized, generalizable learned Transformer-based solvers and current research stage has minimal societal impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All the assets used in the paper are properly licensed. Our paper cites the original papers that produced the two libraries we used:

    - POT 0.9.6.post1: `https://github.com/PythonOT/POT/releases/tag/0.9.6.post1`), distributed under MIT License.
    - Flash Attention 2.8.3: `https://github.com/Dao-AILab/flash-attention`, distributed under BSD 3-Clause License.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Details for the synthetic dataset generation, model architecture and specific code details (e.g. training with data pallalelization, using FlashAttention) are included in the main body of the paper. We will make our code available and provide additional documentation to reproduce our experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification:

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification:

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification:

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

Appendix

Table of Contents

- Appendix A: Experimental Details: ToR-OT Configuration 21
- Appendix B: Runtime comparisons 22
- Appendix C: Post-processing for improved OT feasibility 23
- Appendix D: Mechanistic Interpretability studies 24
- Appendix E: Scaling 26
- Appendix F: Full results on 1D distributions 29
- Appendix G: Full results on 2D distributions 51

A Experimental Details: ToR-OT Configuration

A.1 Training Data (1D)

- **Corpus:** `ivanjaenm/torot_data_5M`; 5M training and 2.5K validation instances
- **Support:** bin centres of a regular grid on $[0, 1]$; $n = m$ uniform in $\{5, \ldots, 100\}$
- **Marginals:** $a, b \sim \mathrm{Dirichlet}(1, \ldots, 1)$, independent
- **Costs:** L_2, $L_2^{1/2}$, L_2^2, L_2^3, Canberra, Gaussian ($\sigma = 0.2$), one per instance; $L_2^p = \|x - y\|_2^p$
- **Targets:** exact plans from POT's network simplex (`ot.emd`)
- **Augmentation:** transpose with prob. 0.5;

A.2 Model

- **Type:** encoder–decoder Transformer
- **Parameters:** 157.6M
- **Width / heads:** 768 / 12 (head dimension 64)
- **Layers:** 6 encoder, 12 decoder
- **Feed-forward:** SwiGLU, hidden width 2,048
- **Normalization:** pre-norm RMSNorm; QK-norm in all attention layers
- **Encoder input:** cost-type token + one token per bin. No positional encoding
- **Decoder:** causal, RoPE, cross-attention to the encoder
- **Column index:** pointer head over the target-bin encoder states

A.3 Tokenization and Loss

- **Row sequence:** `BOS` $\left(\langle j \rangle \langle E_k \rangle d_1 d_2 d_3 d_4\right)^*$ `EOS`, entries in increasing j
- **Mass format:** $0.d_1 d_2 d_3 d_4 \times 10^{-(k-1)}$, $k \in \{1, \ldots, 9\}$ (4 significant digits)
- **Vocabulary:** 23 fixed tokens + one pointer per target bin
- **Loss:** cross-entropy; weight 1 for structure, pointer and exponent tokens, 0.5^{p-1} for mantissa digit p; ordinal label smoothing 0.05 per neighbouring digit; z-loss 10^{-4}

A.4 Optimization

- **Optimizer:** AdamW, $\beta = (0.9, 0.95)$, weight decay 0.1 (weight matrices only)
- **Learning rate:** 10^{-4}, 2,000 warmup steps, cosine decay to 10^{-5}
- **Steps:** 203,000 ($\approx$1.5 passes over the row examples)
- **Batch per GPU:** $\leq$256 rows and $\leq$32,768 encoder tokens; micro-batches of $\leq$24,576 decoder tokens; global $\leq$2,048 rows per step
- **Gradient clipping:** 1.0
- **Precision:** bf16
- **EMA:** 0.999 (EMA weights are evaluated and released)
- **Checkpoint:** lowest validation loss, checked every 5,000 steps (step 200,000)
- **Hardware:** 8$\times$ NVIDIA A100 40 GB, 0.43 s per step, $\approx$24.5 h

A.5 Evaluation

- **Test set:** 1,000 random pairs per cost (training distribution, disjoint seeds)
- **OOD set:** 144 pairs per cost (12 named distributions of the submitted version)
- **Support sizes:** $n = 30$ (main results), 100, 200 (2$\times$ the largest training size); 6,864 instances each
- **2D:** 1,000 test and 49 OOD pairs (7 named distributions) per cost, 6×6 grid
- **Reference plan:** the optimal plan closest to the prediction (LP over the optimal face); equals POT's plan when the optimum is unique

A.6 2D Model

- **Settings:** same architecture, tokenizer, loss and optimizer as above
- **Corpus:** `ivanjaenm/torot_data_2M_2d`; 2M instances on $k \times k$ grids in $[0, 1]^2$, k uniform in $\{3, \ldots, 10\}$ ($n = k^2 \in [9, 100]$); 93.0M row examples
- **Training:** 2$\times$ NVIDIA A100 40 GB, 248,000 steps, $\approx$26 h

B Runtime comparisons

The single-sample runtime comparison favors ToR-OT because its training cost is not included at inference time. To make the comparison more complete, we also report the amortized runtime obtained by spreading the pretraining cost over an increasing number of test samples. The right part of Table 2 shows that the amortized cost decreases rapidly with the number of samples, falling below NeuralOT after as few as 100 samples. The full amortization curve is shown in Figure 6.

Method	Single sample	Amortized analysis		
		100	500	1000
NeuralOT	2627.08	2627.08	2627.08	2627.08
ToR-OT	**1.09**	**2593.18**	**519.60**	**259.57**

Table 2: Runtime comparison (in seconds) of NeuralOT and ToR-OT. Provided an extensive pretraining on multiple OT instances, ToR-OT is much faster in estimating the OT map of a new sample. Additionally, even when we take into account the pretraining stage, the amortized runtime is below NeuralOT as few as 500 samples.

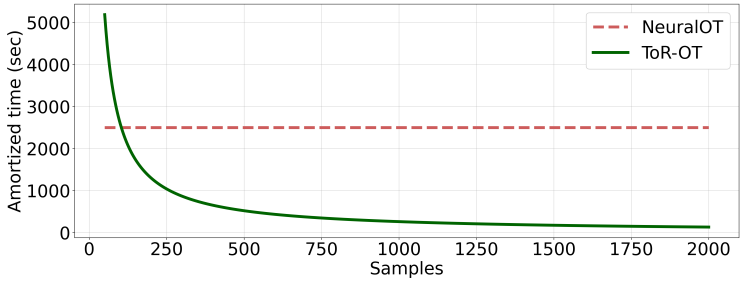

Figure 6: Amortized runtime of ToR-OT taking into account the pretraining time. For as few as 100 samples, ToR-OT drops below NeuralOT, which requires a new training loop for each new sample, with an average runtime of 2600 seconds.

C Post-processing for improved OT feasibility

C.1 Sinkhorn projections

Although ToR-OT produces near-feasible plans (marginal violations of order 10^{-1}–10^{-2} in L_1), applications requiring improved marginal feasibility can use a lightweight post-processing step. We apply a few Sinkhorn iterations to the predicted plan $\hat{\pi}$ to project it onto the feasible set $\Pi(a, b)$ as shown in table Table 3.

Metric	Canonical ToR-OT	**ToR-OT + Sinkhorn**
Max Marginal Violation	1.15×10^{-2}	$\mathbf{1.7 \times 10^{-4}}$
Prediction Error (MAPE)	1.52×10^{-1}	$\mathbf{7.5 \times 10^{-2}}$

Table 3: Improved feasibility and accuracy with a few Sinkhorn iterations on the test set.

Sinkhorn post-processing reduces the maximum marginal violation by nearly two orders of magnitude while simultaneously improving prediction accuracy by approximately 50%. Notably, ToR-OT already outperforms NeuralOT before any post-processing; after post-processing, the solution is both near-feasible to a smaller numerical tolerance and more accurate. This hybrid approach — generate a high-quality sparse plan with ToR-OT, then project to improved feasibility — provides a practical path to certified feasibility without sacrificing the efficiency gains of amortized inference.

C.2 Symmetric decoding via transposed averaging

An additional technique to improve feasibility is symmetric decoding: we run two forward passes with the source and target distributions swapped, then average the resulting plans. This reduces asymmetry between row and column marginal residuals as Table 4 shows.

Metric	Canonical (Single Pass)	**Symmetric (Two Pass Avg.)**
Max Marginal Violation	0.0272	**0.0218**
Prediction Error (MAPE)	0.3025	**0.2538**
Total Inference Time	1.133s	**2.299**s

Table 4: Effect of symmetric two-pass averaging on feasibility, accuracy, and inference time.

Symmetric decoding reduces asymmetry by $\sim 20\%$ and improves accuracy by $\sim 16\%$, at the cost of doubling inference time.

D Mechanistic Interpretability studies

We report three mechanistic studies designed to distinguish whether ToR-OT has internalized the mathematical structure of OT. These studies were motivated by methods from the mechanistic interpretability literature Bietti et al. [2023], Olsson et al. [2022].

D.1 Causal Intervention (inspired by induction head analysis)

For a target row i, we modify the serialized prefix of previous rows to simulate exhausting column j's global capacity: we set the cumulative allocation $\sum_{i'<i} \pi_{i'j}$ equal to b_j, so no mass remains available for column j. This is done without changing the global problem instance (μ, ν, C) or any model weights, isolating the causal effect of the prefix state on generation. Across 100 such interventions, the model adapted its routing in **99%** of cases: mean mass reduction to column j was **87.3%**, with **42%** of interventions showing complete avoidance (mass $\to 0$) and the remaining **57%** showing proportional reduction. The two modes of response indicate the model tracks remaining capacity as a *continuous* quantity, not a binary flag. Since the only change was in the prefix state while the global instance remained fixed, this is direct causal evidence that ToR-OT conditions generation on the evolving global constraint state — not on shallow distributional patterns from the input.

D.2 Linear Probing of Residual Streams

We extracted final-layer hidden states immediately before each mass prediction and trained linear probes (ridge regression) to recover the remaining column capacity vector $r_j = b_j - \sum_{i'<i} \pi_{i'j} \in \mathbb{R}^{30}$ on held-out sequences. Ridge regression achieves $R^2 = \mathbf{0.7135}$, showing that the remaining constraint budget is *linearly accessible* from the residual stream. The probing accuracy increases monotonically as generation progresses within a row, consistent with two-timescale learning dynamics Bietti et al. [2023]: the model dynamically updates its representation of the constraint state rather than encoding it statically from the input. This complements the causal intervention: the probing shows *that* the constraint state is encoded; the intervention shows *that it is used*.

D.3 Attention Pattern Analysis

We visualized last-layer self-attention during generation over 144 attention maps, grouping character-level tokens into three interpretable blocks: ***Prefix*** (encoding a, b, C, i/n), ***Index tokens*** (j_k), and ***Weight tokens*** (w_k). Three consistent patterns appear across maps: (i) **Global heads** — several heads maintain persistent attention to the Prefix block throughout decoding, providing continuous reference to global marginal constraints, analogous to the global heads identified in Bietti et al. [2023]. (ii) **Running accumulator heads** — in longer rows, specific heads show weight-to-weight off-diagonal bands, attending to previously emitted mass tokens when computing subsequent weights, consistent with a running accumulator tracking the remaining row budget. (iii) **Spatial allocation heads** — tight index-to-index staircase patterns suggest local spatial boundary enforcement under the cost geometry. These three head types coordinate simultaneously across global constraint tracking, accumulated mass tracking, and spatial allocation, inconsistent with independent per-token prediction and supporting the interpretation that the model has learned a structured internal algorithm for OT plan generation.

D.4 Why Autoregressive Decoding for OT?

The convergent evidence above raises a structural question: why is autoregressive decoding specifically well-suited to OT, rather than, say, a fixed-size dense output head? Two properties of OT's mathematical structure suggest a natural alignment.

First, OT feasibility constraints are *globally coupled*: each entry π_{ij} constrains all others sharing row i or column j. The serialized prefix $\pi_{<i,:}$ encodes accumulated column allocations, making the remaining capacity $b_j - \sum_{i'<i} \pi_{i'j}$ implicitly computable from context — without any explicit constraint enforcement mechanism. This is precisely what the causal intervention and linear probing results confirm: the model tracks this evolving capacity state in its hidden representations and uses it causally during generation.

Second, the polyhedral structure of the Kantorovich LP guarantees that optimal plans have *at most* $n + m - 1$ nonzero entries, but the exact support is instance-dependent and varies in size.

Autoregressive generation with an end-of-sequence token handles this variable-length structured output naturally, whereas a fixed-size dense prediction head must either over-allocate capacity or impose an arbitrary sparsification threshold post hoc. The three head types observed in the attention analysis: global constraint heads, running accumulators, and spatial allocation heads, correspond directly to the three computational demands this structure imposes: track global budgets, accumulate row mass, and resolve local cost geometry.

Together, these observations suggest that the strong empirical performance of ToR-OT is not incidental: structural alignments between the autoregressive mechanism and the mathematical properties of OT make this an effective inductive bias for learning to predict sparse transport plans.

E Scaling

This appendix details the experiments on problem and training scale: support sizes beyond the main setting (E.1), training length (E.2) and, on a smaller testbed, corpus size, model size and design choices (E.3).

E.1 Support size

The 1D model is trained on supports of 5 to 100 bins. We evaluate it on test and OOD sets built as in the main experiments at $n = m = 100$, inside the training range, and at $n = m = 200$, twice the largest training size; Meta OT and UNOT are retrained at every n (three seeds each). Table 5 compares ToR-OT with Meta OT, the strongest baseline, each scored against the solution of the problem it solves.

On the test set, ToR-OT's error grows moderately from $n = 30$ to $n = 100$ (0.009–0.064 to 0.027–0.062), while Meta OT's grows several-fold (0.072–0.221 to 0.28–0.73), so the gap widens with n. On the OOD set at $n = 100$, ToR-OT remains more accurate on most costs, but falls behind Meta OT on L_2^2 and L_2^3. At $n = 200$ the model does not extrapolate for L_2^2, L_2^3 and Gaussian: its plans are far from optimal (MAPE above 1). For L_2, $L_2^{1/2}$ and Canberra, the error grows but stays below Meta OT's. Supports larger than those seen in training therefore require training on them.

The plan length grows linearly with n (386 tokens at $n = 30$, 1,296 at $n = 100$ and 2,596 at $n = 200$; log–log slope 1.01), and so does the time to generate one plan on one GPU (2.5 s, 8.6 s and 18.3 s; slope 1.03).

Table 5: Mean MAPE at three support sizes, each method against the solution of the problem it solves (lower is better; bold: the better of the two). ToR-OT is the single 1D model trained on $n \leq 100$; Meta OT is retrained at every n (mean over three training seeds). Test: 1,000 pairs per cost; OOD: the 144 pairs of the 12 named distributions, binned at each n.

		$n = 30$		$n = 100$		$n = 200$	
Set	Cost	ToR-OT	Meta OT	ToR-OT	Meta OT	ToR-OT	Meta OT
Test	L_2	**0.009**	0.072	**0.027**	0.436	**0.302**	0.701
	$L_2^{1/2}$	**0.016**	0.180	**0.037**	0.282	**0.375**	0.395
	L_2^2	**0.010**	0.221	**0.031**	0.612	1.770	**0.804**
	L_2^3	**0.010**	0.148	**0.031**	0.581	1.767	**0.733**
	Canberra	**0.013**	0.154	**0.032**	0.733	**0.371**	0.945
	Gaussian	**0.064**	0.176	**0.062**	0.733	1.727	**0.326**
OOD	L_2	**0.031**	0.078	**0.090**	0.438	**0.203**	0.448
	$L_2^{1/2}$	**0.037**	0.104	**0.101**	0.200	**0.232**	0.289
	L_2^2	**0.056**	0.229	0.319	**0.301**	1.326	**0.486**
	L_2^3	**0.057**	0.175	0.318	**0.247**	1.347	**0.286**
	Canberra	**0.039**	0.127	**0.099**	0.651	**0.227**	0.755
	Gaussian	**0.062**	0.200	**0.313**	0.714	1.348	**0.338**

E.2 Training length

To trace how accuracy depends on the amount of training data seen, at no additional training cost, we evaluate eight checkpoints of the 1D model, every 25,000 steps from 47M to 374M training rows (≈1.5 passes over the corpus; Fig. 7). They are evaluated on fixed subsets of the corpus' own test splits: 1,000 random pairs (n from 5 to 100) and its 864 named-distribution pairs (144 per cost, $n = 30$), with MAPE against POT's plan, computed during evaluation (for costs with tied optima, such as L_2, this overstates the error).

The error on the named-distribution pairs decreases steadily, from 0.072 to 0.031 (median), and is still decreasing at the last checkpoint, as are the support recall and the marginal residuals, so longer training would likely help further. The in-distribution error decreases faster (0.063 to 0.017), so the ratio of OOD to test error grows from 1.1 to 1.8. All checkpoints come from one run, so the number of rows seen and the learning-rate schedule change together; the effect of the corpus itself is isolated in E.3.

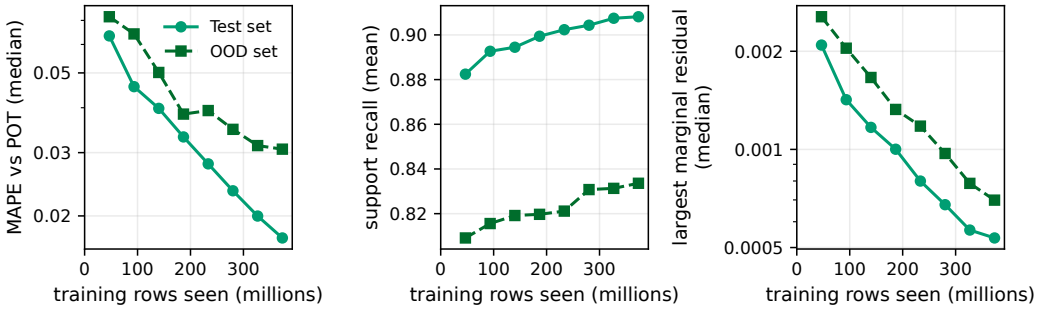

Figure 7: Checkpoints of the 1D model (every 25,000 steps) vs training rows seen: median MAPE against POT's plan, mean support recall and median largest marginal residual, on 1,000 test pairs and the 864 named-distribution pairs of the corpus' OOD split.

E.3 Corpus size, model size and design choices

Retraining the full model for every configuration is too costly, so these ablations use a smaller testbed with the same architecture, tokenizer and loss: a 9.8M-parameter model (width 256, 4 heads, 4 encoder and 6 decoder layers) trained on the 270,508 instances with $n \leq 30$ of a 1M-instance corpus (4.5M row examples). Every run has the same training budget, 11,000 steps of 2,048 rows (22.5M rows, about one GPU-hour), and every configuration is trained with three seeds. The models are evaluated on 1,311 test pairs and 864 named-distribution pairs ($n = 30$), with MAPE against POT's plan (Fig. 8, Table 6).

Corpus size. With the training budget fixed, the model trained on 10% of the instances passes over them 50 times, against 5 times for the full corpus. Its test and OOD errors are nevertheless the same within the variation across seeds (OOD: 0.089–0.132 for 10% to 100%; the three seeds of the base recipe alone span 0.079–0.132). At this scale, the model does not overfit the training instances even after many passes, and accuracy on unseen distributions does not depend on a large pool of distinct instances.

Model size. Capacity matters more: the OOD error falls from 0.36 to 0.10 as the model grows from 0.7M to 39.3M parameters, and the test error from 0.068 to 0.034, with smaller gains beyond 9.8M at this training length.

Design choices. Perturbing the remaining capacities during training (App. A) matters most: without it, the test error rises from 0.036 to 0.047 and the OOD error from 0.113 to 0.165, as the model no longer sees the slightly inexact capacities it produces at inference. The transpose augmentation and the digit-weighted loss change the errors by less than the variation across seeds.

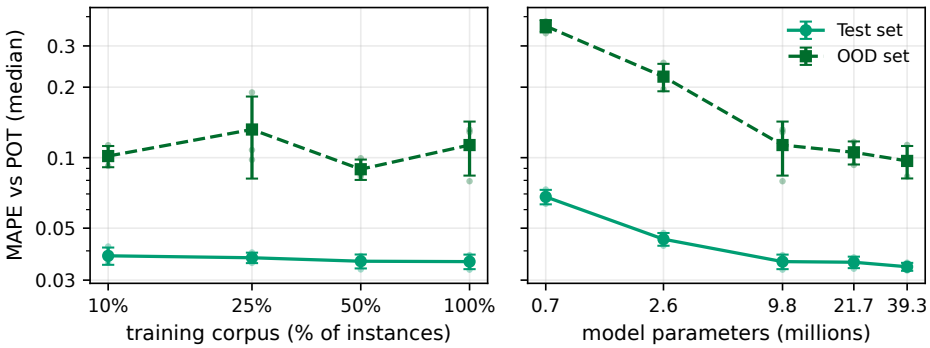

Figure 8: Small testbed ($n \leq 30$, the same training budget for every run): median MAPE against POT's plan vs the fraction of training instances (left) and the number of parameters (right). Line: mean over three training seeds; error bar: standard deviation; dots: the seeds.

Table 6: Ablations on the small testbed: median MAPE against POT's plan, mean $\pm$ standard deviation over three training seeds. Each row changes one aspect of the base recipe.

	Configuration	Parameters	Test set	OOD set
Base recipe	width 256 (base)	9.8M	0.036 ± 0.003	0.113 ± 0.029
Model size	width 64	0.7M	0.068 ± 0.005	0.365 ± 0.022
	width 128	2.6M	0.045 ± 0.003	0.222 ± 0.030
	width 384	21.7M	0.036 ± 0.002	0.105 ± 0.012
	width 512	39.3M	0.034 ± 0.001	0.097 ± 0.015
Training corpus	10% of the instances	9.8M	0.038 ± 0.003	0.102 ± 0.011
	25% of the instances	9.8M	0.037 ± 0.002	0.132 ± 0.050
	50% of the instances	9.8M	0.036 ± 0.002	0.089 ± 0.009
Design choices	no capacity perturbation	9.8M	0.047 ± 0.007	0.165 ± 0.069
	no transpose augmentation	9.8M	0.036 ± 0.002	0.142 ± 0.083
	plain cross-entropy	9.8M	0.040 ± 0.003	0.107 ± 0.012

F Full results on 1D distributions

Contents

F.1 Cost type: L_2 29

F.2 Cost type: $L_2^{1/2}$ 33

F.3 Cost type: L_2^2 36

F.4 Cost type: L_2^3 39

F.5 Cost type: Canberra 42

F.6 Cost type: Gaussian 45

F.7 All six cost types 48

This appendix reports every method on every cost at $n = m = 30$, on the two sets of Fig. 3:

- **Test set:** 1K random pairs per cost, marginals uniform on the simplex (training distribution).
- **OOD set:** the 144 ordered pairs of the 12 named distributions.

As in Fig. 3, each method is scored against the solution of the problem it solves. ToR-OT and NeuralOT (exact OT) are compared with the optimal plan closest to their prediction, found by a linear program over the optimal face (POT's plan whenever the optimum is unique). Meta OT and UNOT (entropic OT) are compared with the converged Sinkhorn plan at their ε, given in each subsection. The metrics, for a prediction $\hat{\pi}$ and reference π:

- **MAPE:** $\sum |\hat{\pi}_{ij} - \pi_{ij}| / \sum \pi_{ij}$ over the union of the two supports (the main metric).
- **MAE:** mean of $|\hat{\pi}_{ij} - \pi_{ij}|$ over the entries where the two plans differ. An entropic plan differs from its reference by small amounts on almost every entry, which lowers its MAE.
- **Hellinger:** $\frac{1}{\sqrt{2}} \|\sqrt{\hat{\pi}} - \sqrt{\pi}\|_2$.
- **TV:** $\frac{1}{2} \sum |\hat{\pi}_{ij} - \pi_{ij}|$.
- **|OT cost diff.|:** $|\langle C, \hat{\pi} \rangle - \langle C, \pi \rangle|$.

The box plots use training seed 0 of the learned baselines; the numbers in the text are the means of Fig. 3 (three seeds for the baselines). Each subsection has a summary figure and the plans of the 144 OOD pairs for POT and each method. In these plan grids, the panel in row i and column j is the 30×30 plan from source distribution i to target distribution j, with its own color scale from 0 to its largest entry; Meta OT and UNOT solve entropic OT, so their plans are blurred by design. Sec. F.7 compares all six costs.

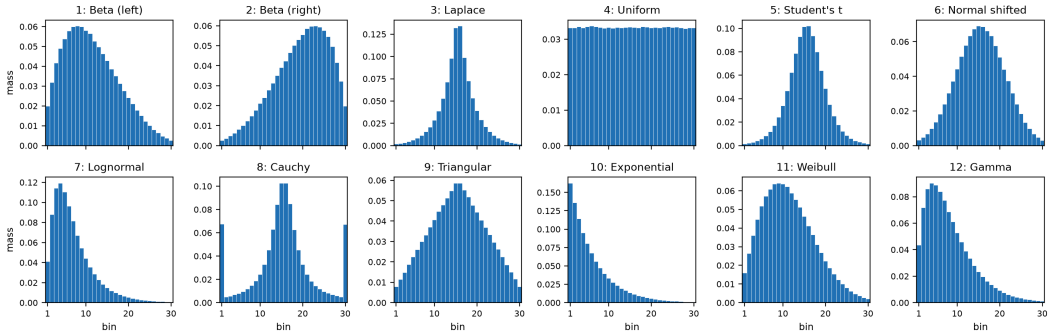

Figure 9: The 12 distributions of the 1D OOD set, numbered as in the improvement heatmaps (Fig. 48).

F.1 Cost type: L_2

Cost $c(x, y) = |x - y|$; Meta OT uses $\varepsilon = 0.01$ and UNOT $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.009 on the test set and 0.031 on the OOD set, against 0.072 and 0.078 for Meta OT, the strongest

baseline (Fig. 10). In 1D the optimal plan for L_2 is often not unique: POT's plan (Fig. 11) is one of several, and ToR-OT and NeuralOT are scored against the optimal plan closest to their prediction.

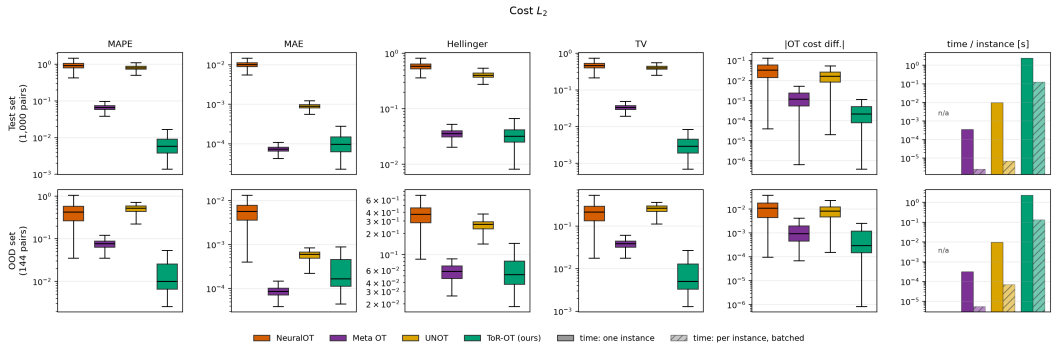

Figure 10: Cost L_2: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

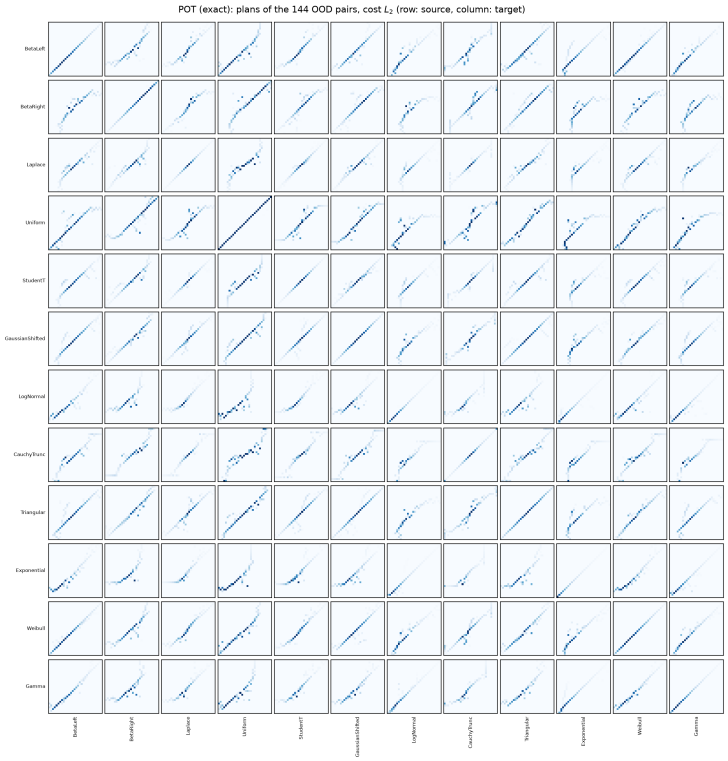

Figure 11: POT (exact): plans of the 144 OOD pairs, cost L_2.

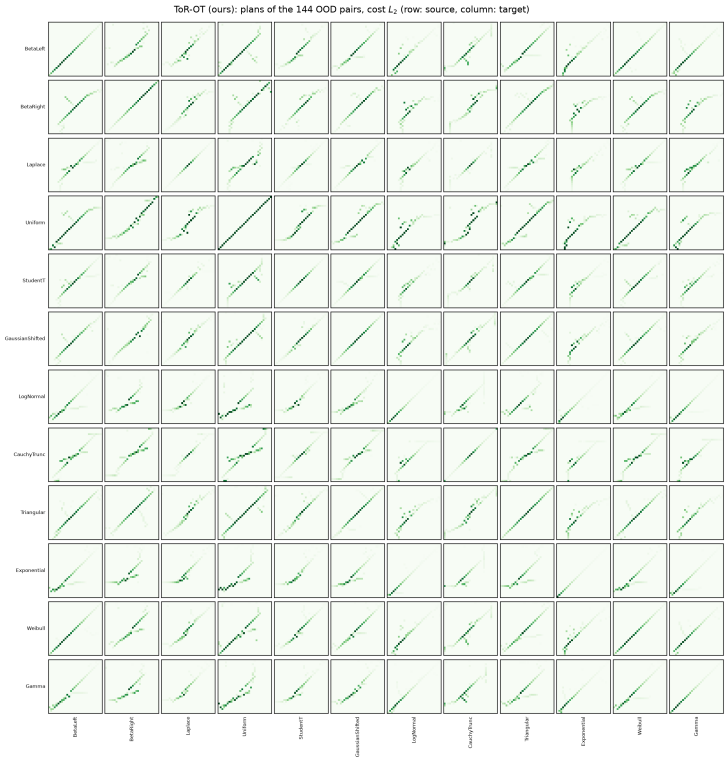

Figure 12: ToR-OT (ours): plans of the 144 OOD pairs, cost L_2.

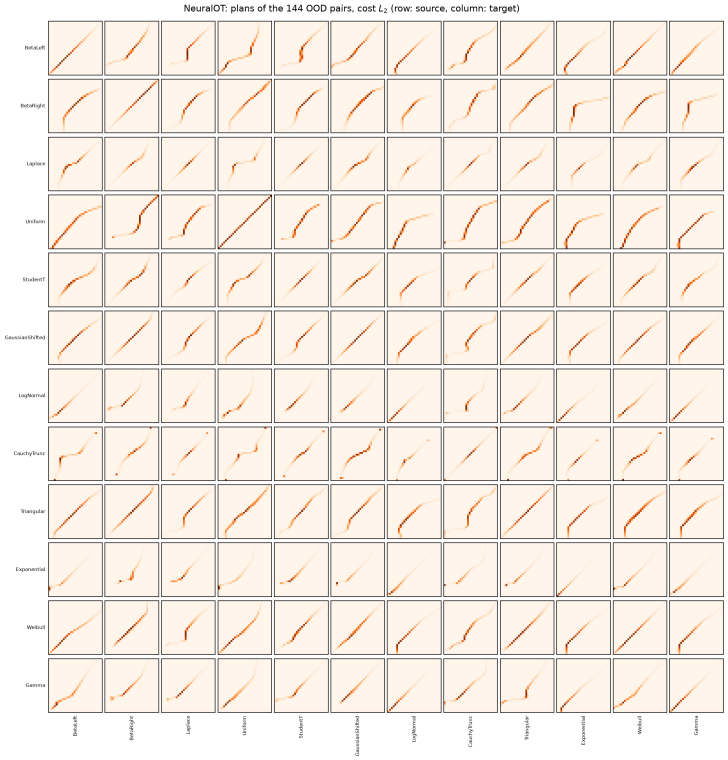

Figure 13: NeuralOT: plans of the 144 OOD pairs, cost L_2.

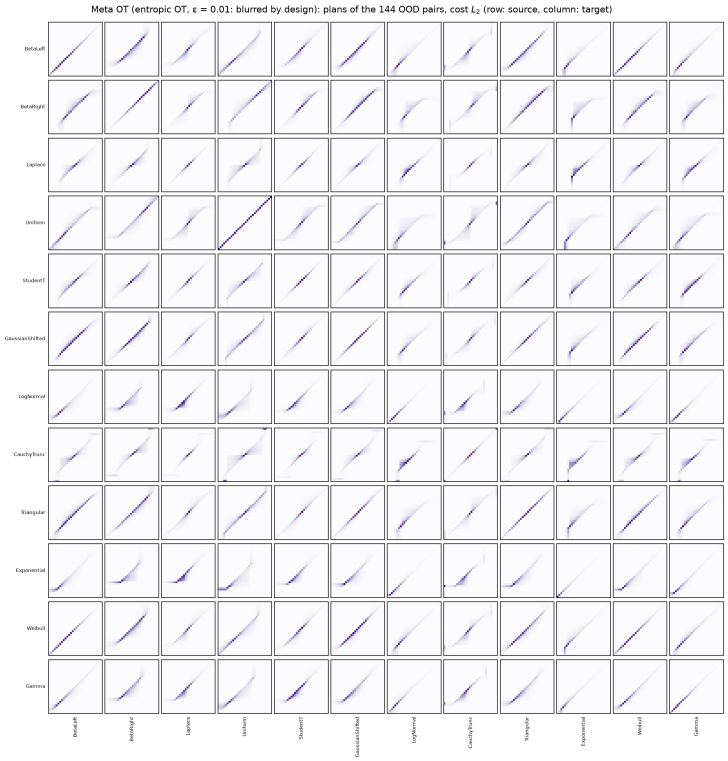

Figure 14: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost L_2.

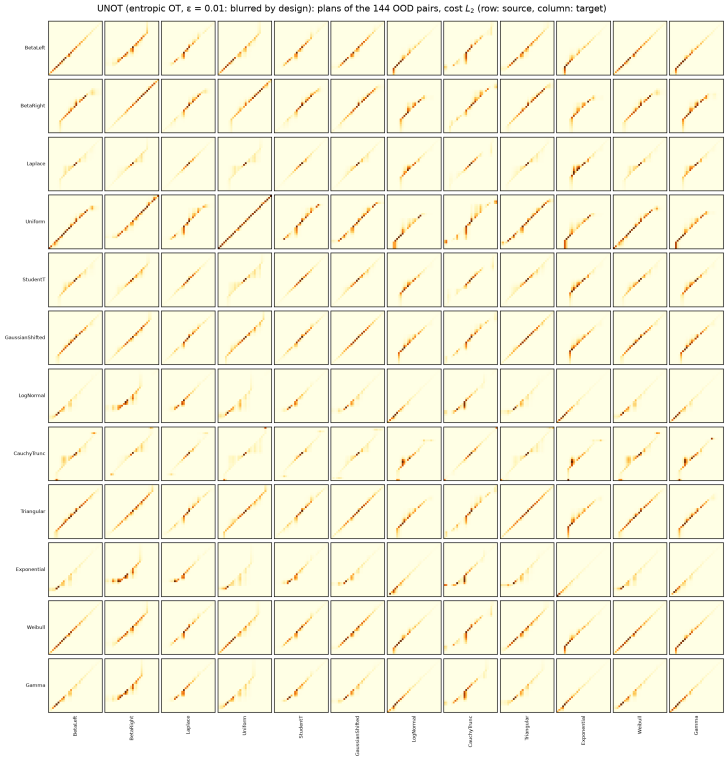

Figure 15: UNOT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost L_2.

F.2 Cost type: $L_2^{1/2}$

Cost $c(x, y) = |x - y|^{1/2}$ (concave); Meta OT uses $\varepsilon = 0.01$ and UNOT $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.016 on the test set and 0.037 on the OOD set, against 0.180 and 0.104 for Meta OT, the strongest baseline (Fig. 16).

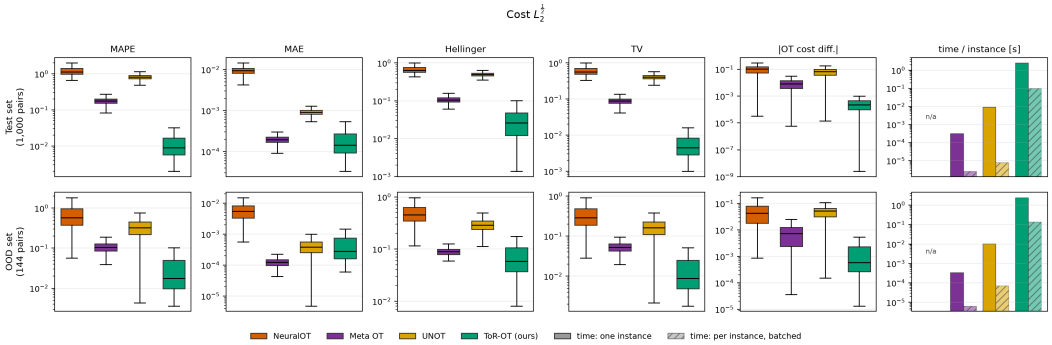

Figure 16: Cost $L_2^{1/2}$: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

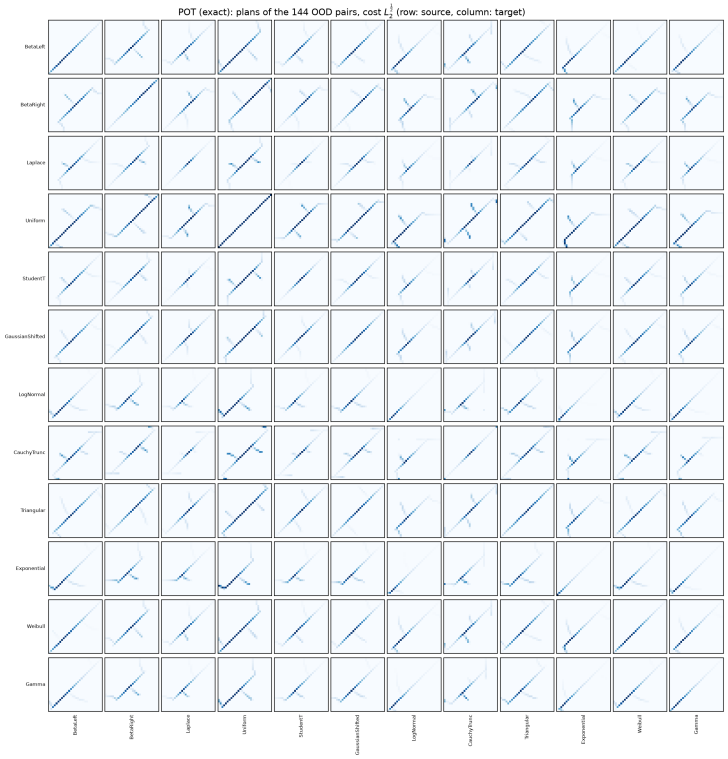

Figure 17: POT (exact): plans of the 144 OOD pairs, cost $L_2^{1/2}$.

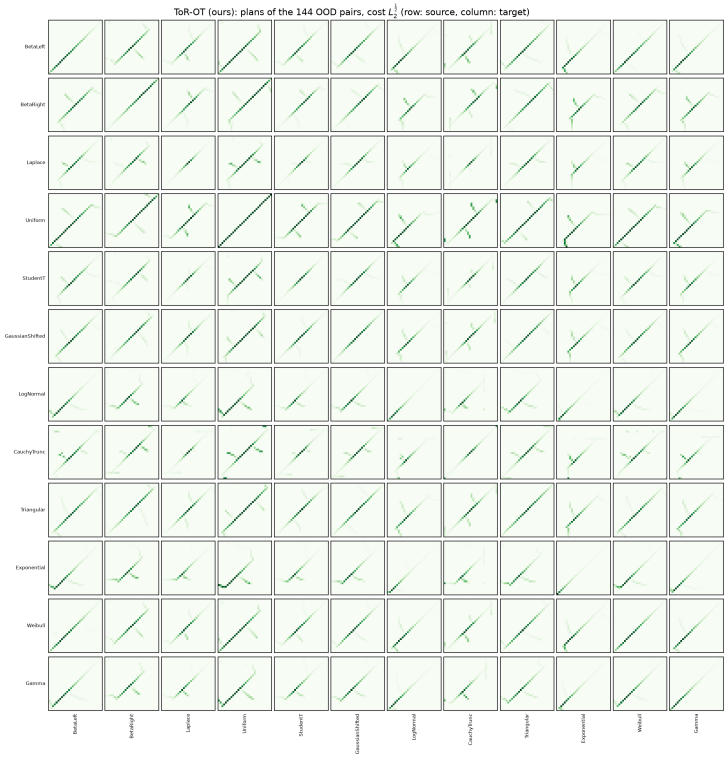

Figure 18: ToR-OT (ours): plans of the 144 OOD pairs, cost $L_2^{1/2}$.

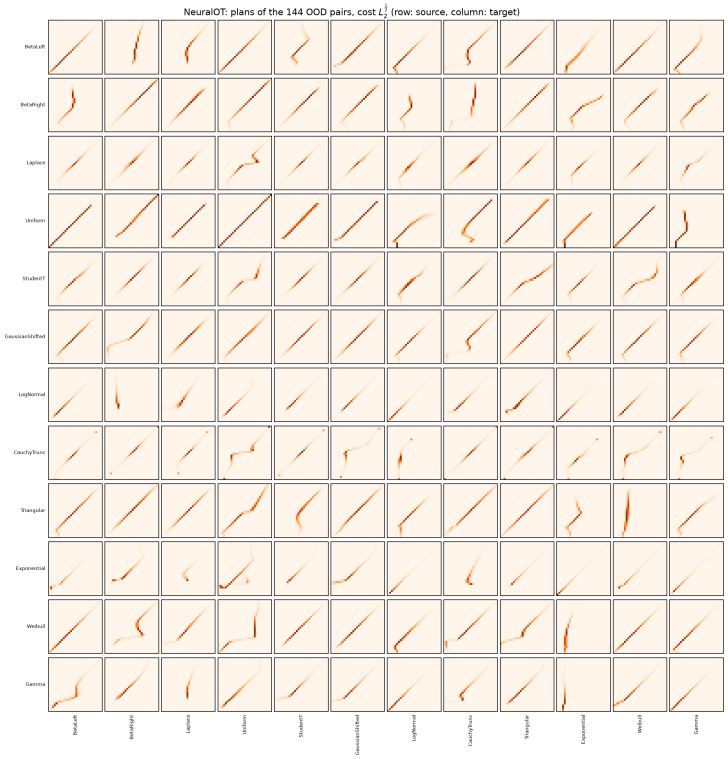

Figure 19: NeuralOT: plans of the 144 OOD pairs, cost $L_2^{1/2}$.

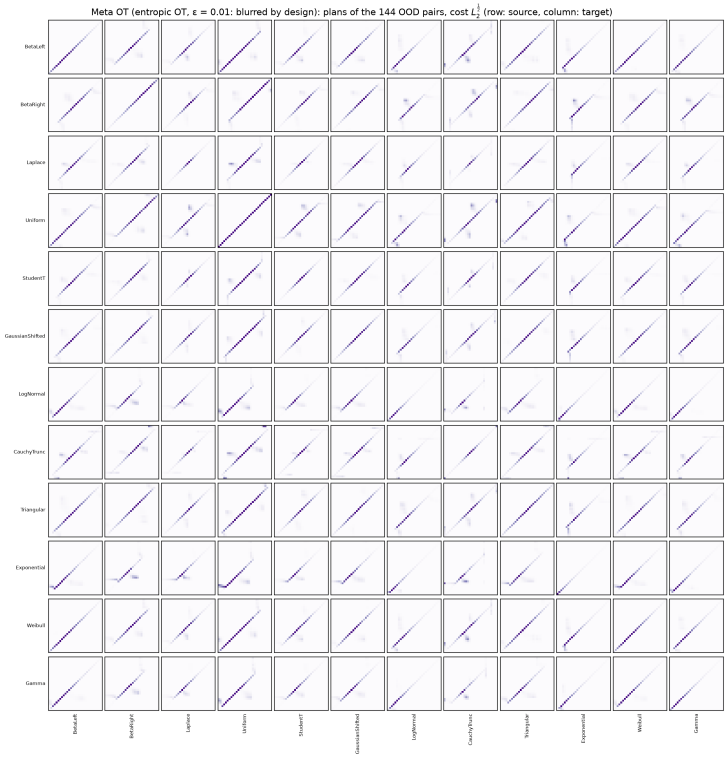

Figure 20: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost $L_2^{1/2}$.

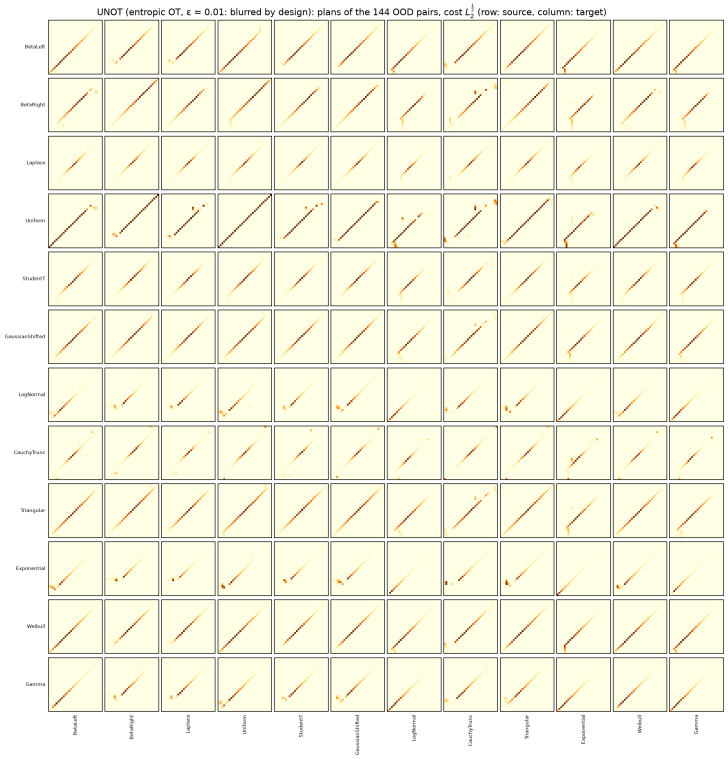

Figure 21: UNOT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost $L_2^{1/2}$.

F.3 Cost type: L_2^2

Cost $c(x, y) = |x - y|^2$; Meta OT uses $\varepsilon = 0.001$ and UNOT $\varepsilon = 0.001$. ToR-OT's mean MAPE is 0.010 on the test set and 0.056 on the OOD set, against 0.221 and 0.229 for Meta OT, the strongest baseline (Fig. 22).

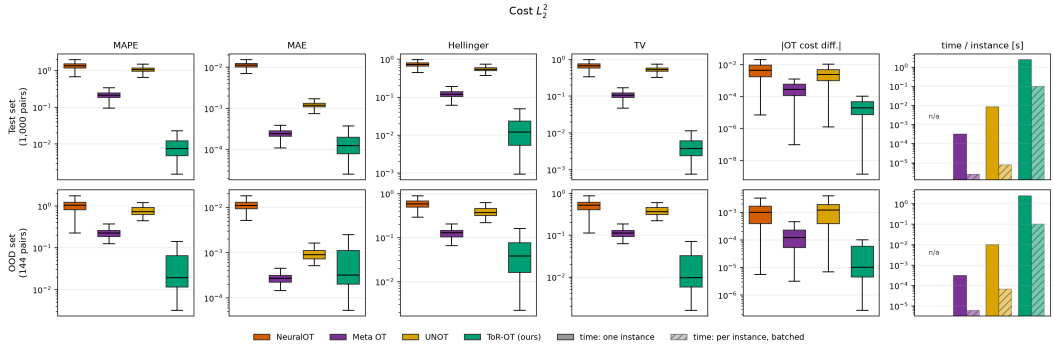

Figure 22: Cost L_2^2: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

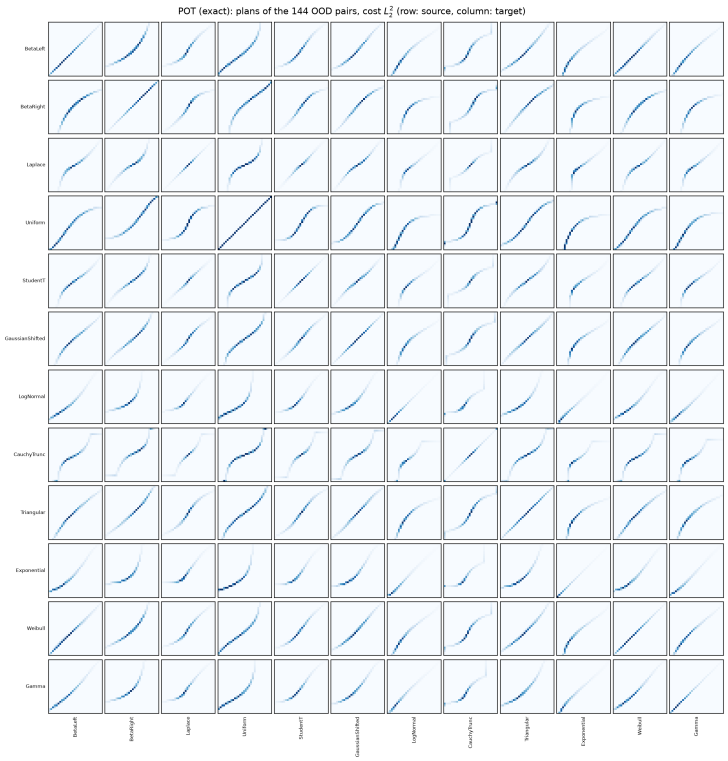

Figure 23: POT (exact): plans of the 144 OOD pairs, cost L_2^2.

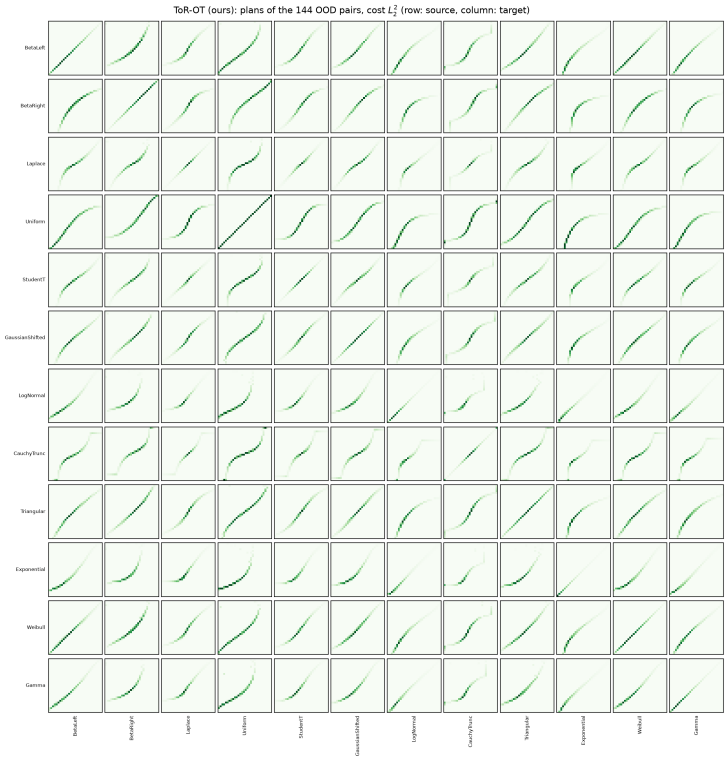

Figure 24: ToR-OT (ours): plans of the 144 OOD pairs, cost L_2^2.

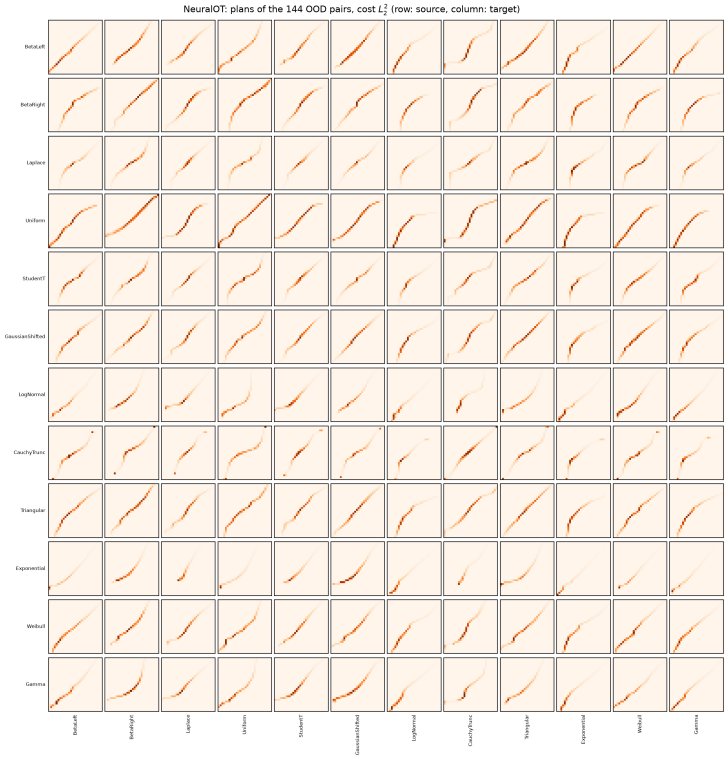

Figure 25: NeuralOT: plans of the 144 OOD pairs, cost L_2^2.

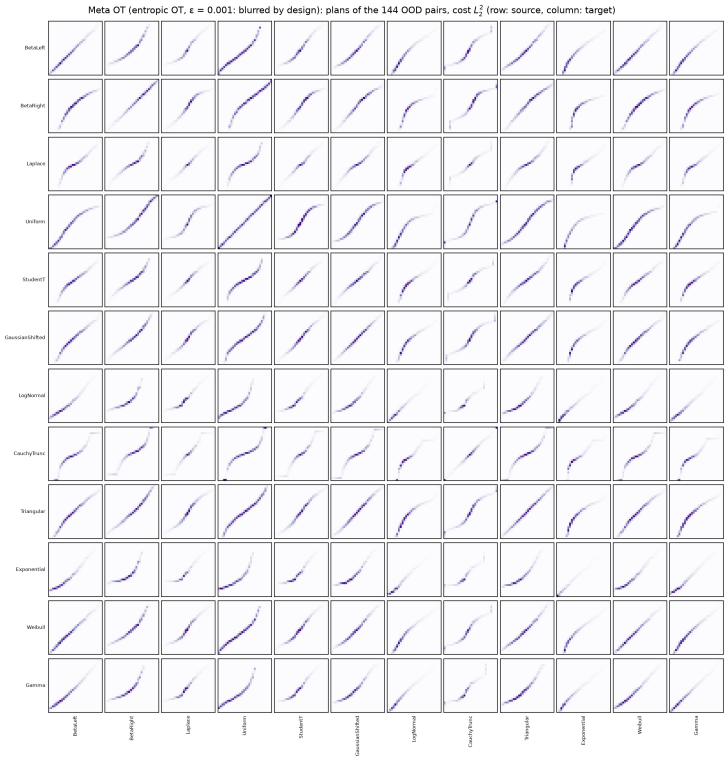

Figure 26: Meta OT (entropic, $\varepsilon = 0.001$): plans of the 144 OOD pairs, cost L_2^2.

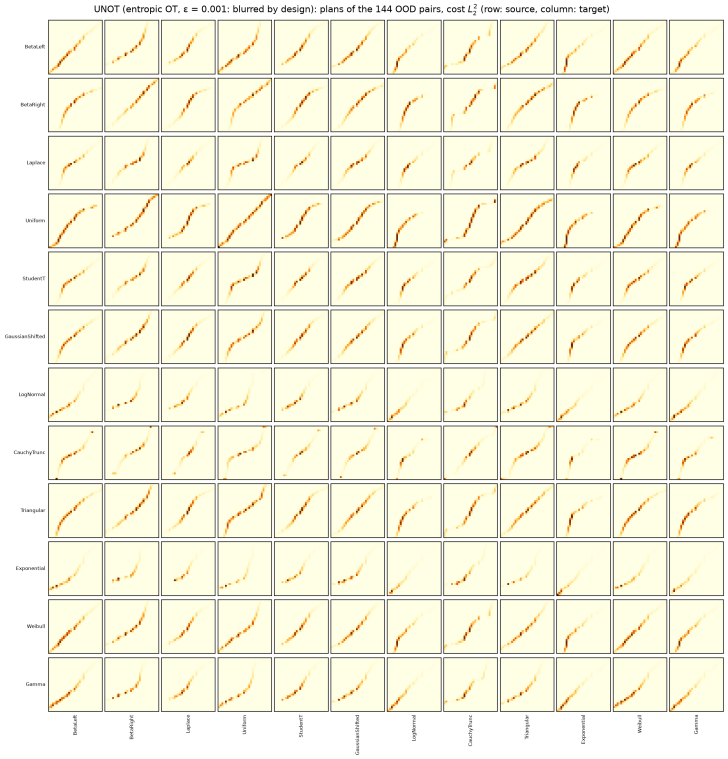

Figure 27: UNOT (entropic, $\varepsilon = 0.001$): plans of the 144 OOD pairs, cost L_2^2.

F.4 Cost type: L_2^3

Cost $c(x, y) = |x - y|^3$; Meta OT uses $\varepsilon = 0.001$ and UNOT $\varepsilon = 0.001$. ToR-OT's mean MAPE is 0.010 on the test set and 0.057 on the OOD set, against 0.148 and 0.175 for Meta OT, the strongest baseline (Fig. 28).

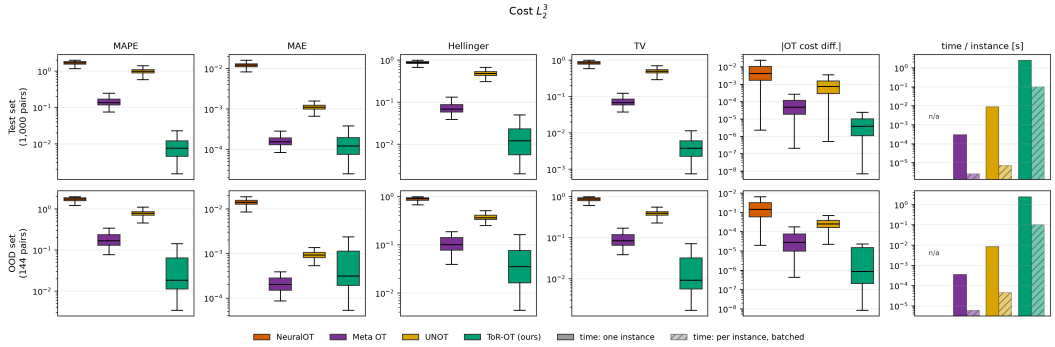

Figure 28: Cost L_2^3: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

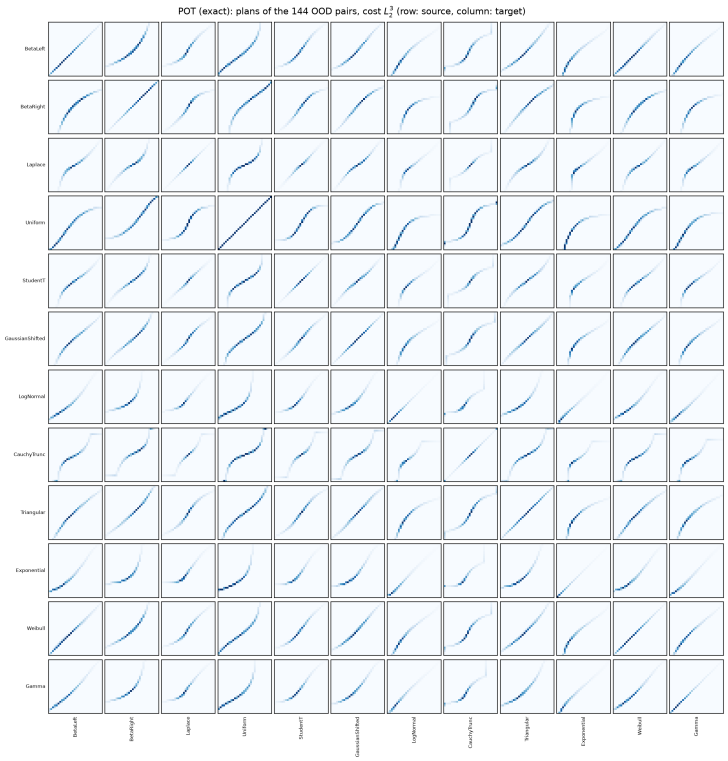

Figure 29: POT (exact): plans of the 144 OOD pairs, cost L_2^3.

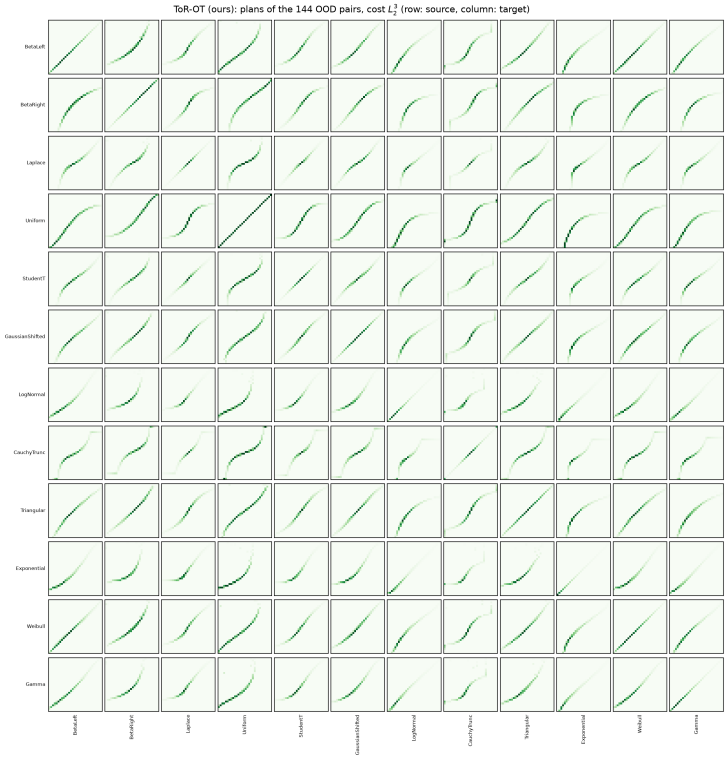

Figure 30: ToR-OT (ours): plans of the 144 OOD pairs, cost L_2^3.

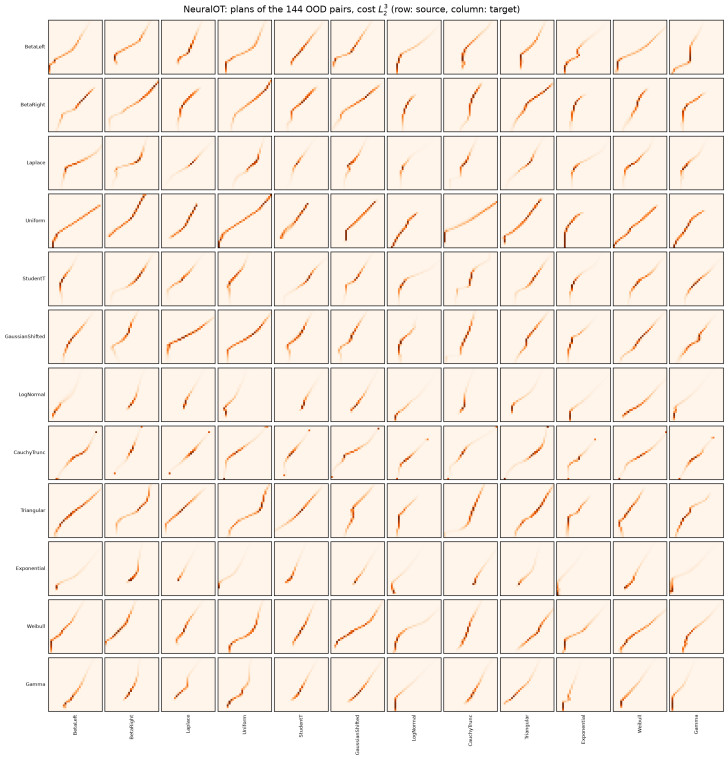

Figure 31: NeuralOT: plans of the 144 OOD pairs, cost L_2^3.

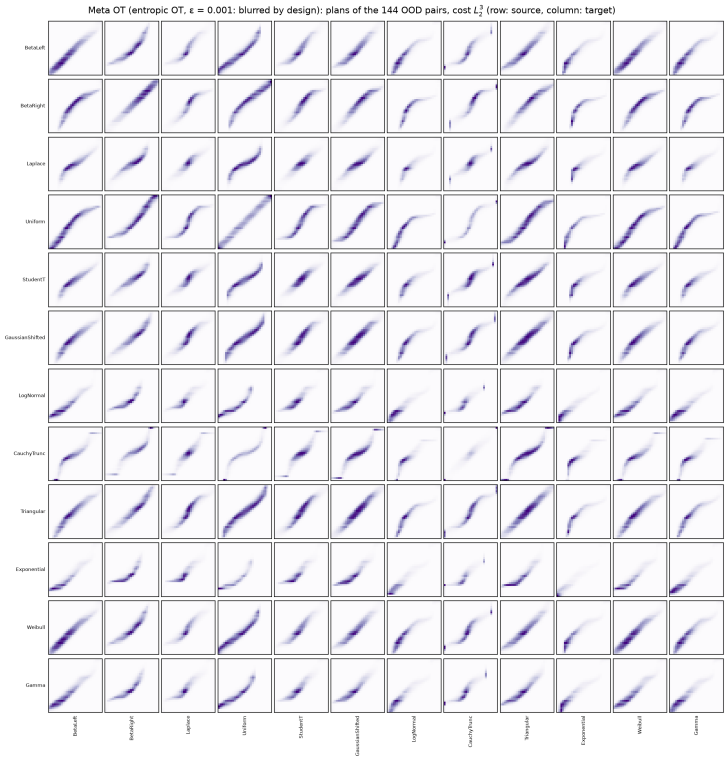

Figure 32: Meta OT (entropic, $\varepsilon = 0.001$): plans of the 144 OOD pairs, cost L_2^3.

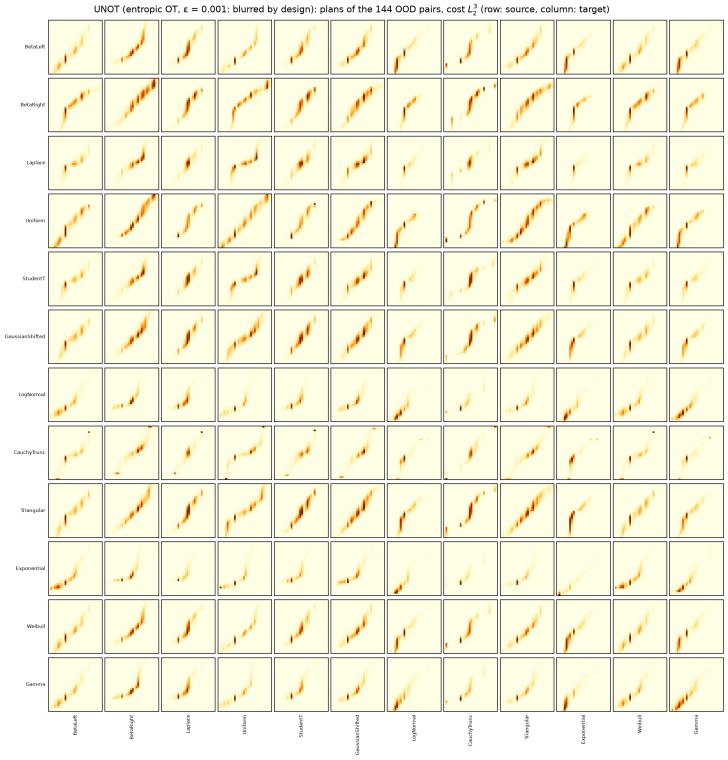

Figure 33: UNOT (entropic, $\varepsilon = 0.001$): plans of the 144 OOD pairs, cost L_2^3.

F.5 Cost type: Canberra

Cost $c(x, y) = |x - y|/(|x| + |y|)$; Meta OT uses $\varepsilon = 0.01$ and UNOT $\varepsilon = 0.001$. ToR-OT's mean MAPE is 0.013 on the test set and 0.039 on the OOD set, against 0.154 and 0.127 for Meta OT, the strongest baseline (Fig. 34).

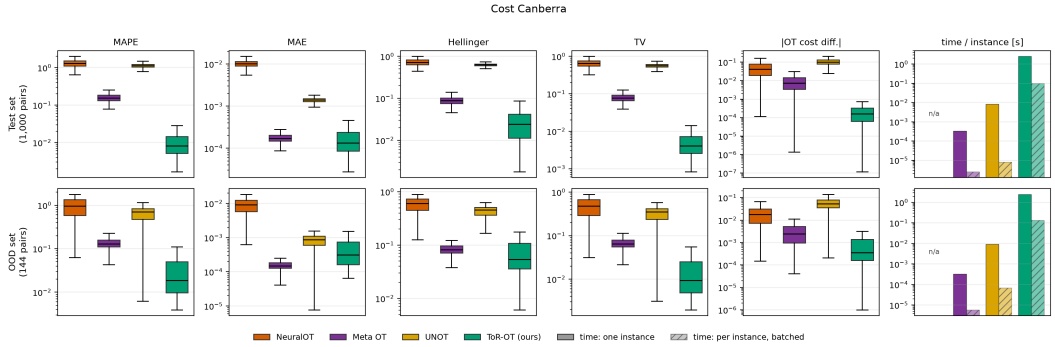

Figure 34: Cost Canberra: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

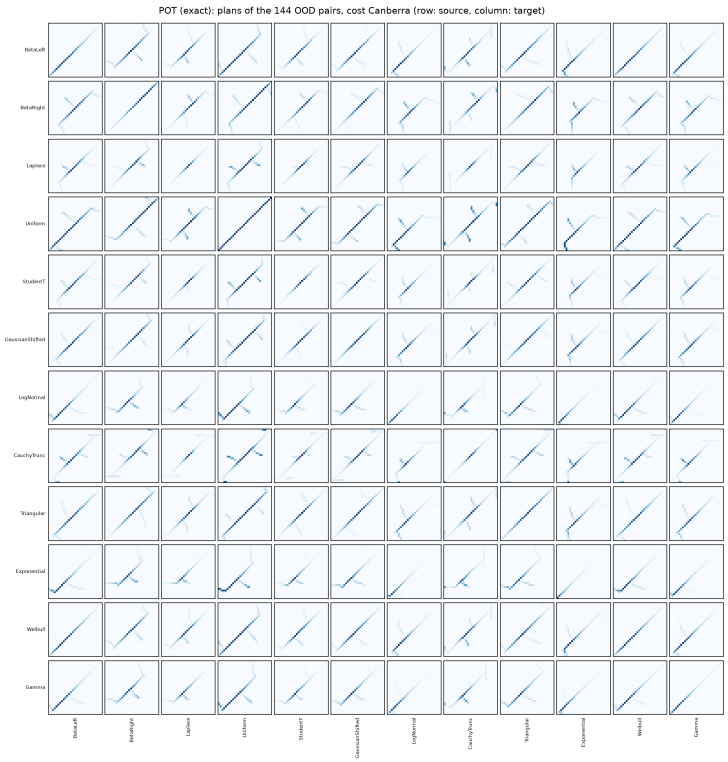

Figure 35: POT (exact): plans of the 144 OOD pairs, cost Canberra.

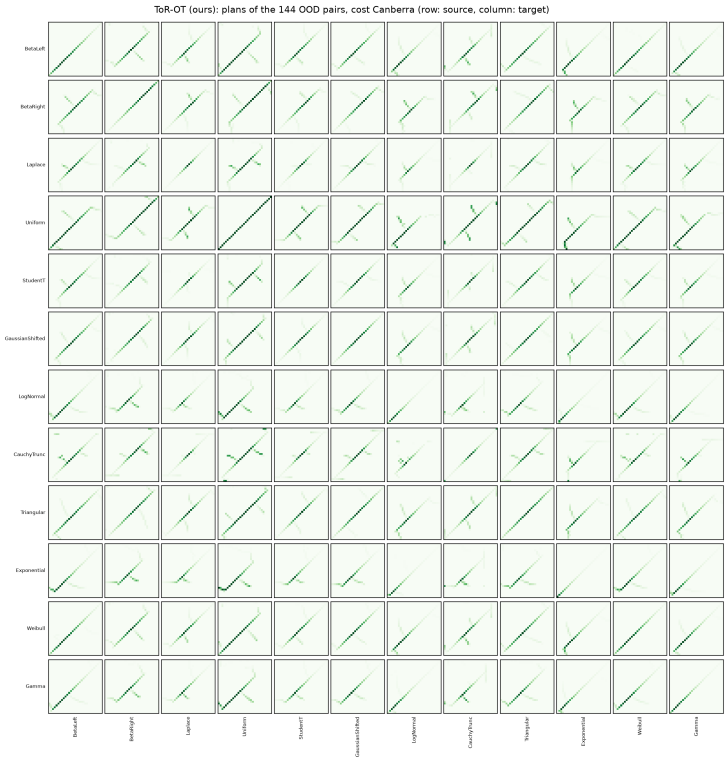

Figure 36: ToR-OT (ours): plans of the 144 OOD pairs, cost Canberra.

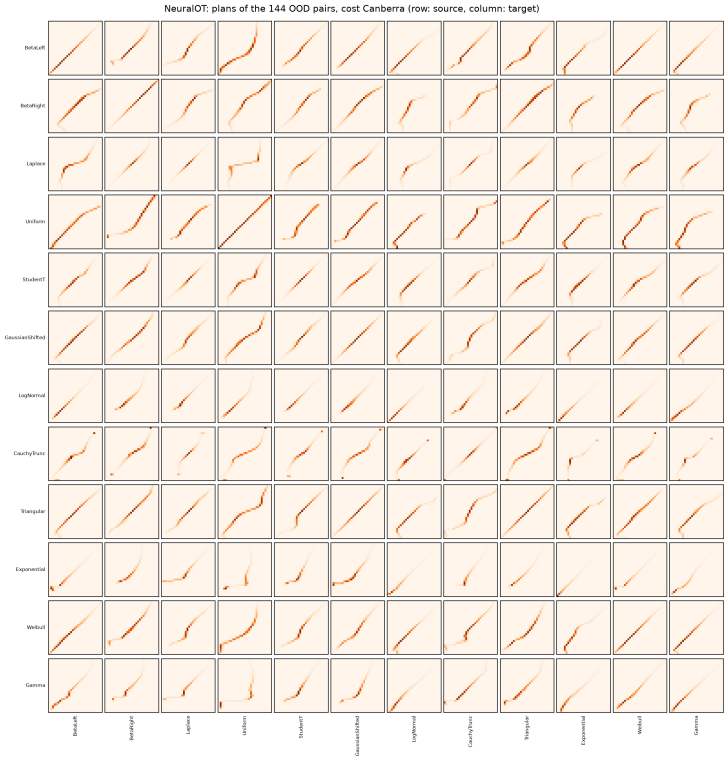

Figure 37: NeuralOT: plans of the 144 OOD pairs, cost Canberra.

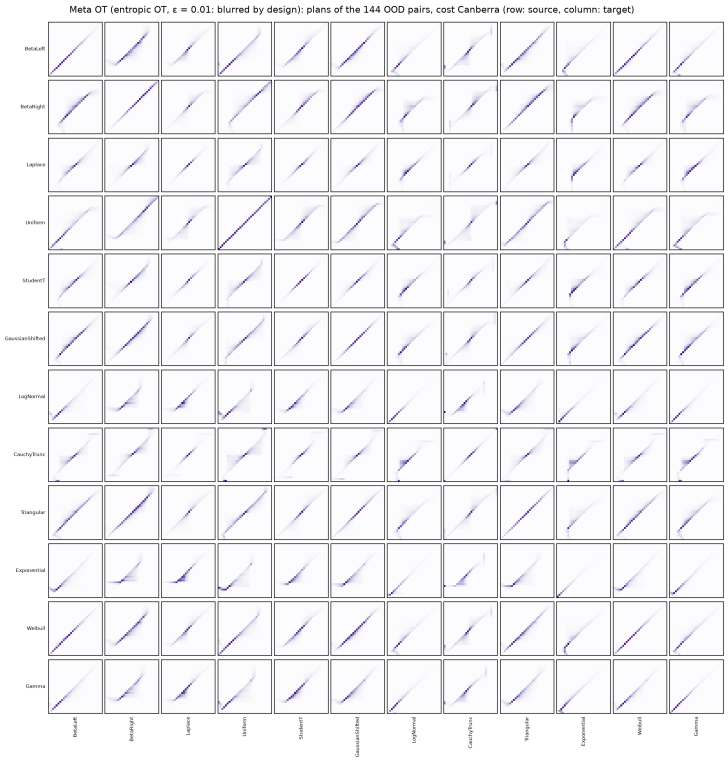

Figure 38: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost Canberra.

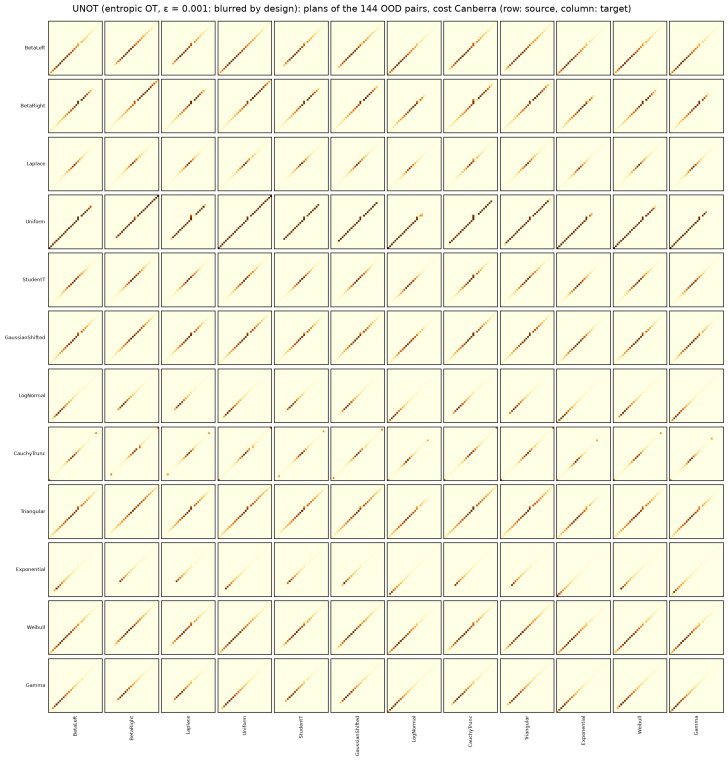

Figure 39: UNOT (entropic, $\varepsilon = 0.001$): plans of the 144 OOD pairs, cost Canberra.

F.6 Cost type: Gaussian

Cost $c(x, y) = 1 - \exp(-|x - y|^2/2\sigma^2)$, $\sigma = 0.2$; Meta OT uses $\varepsilon = 0.01$ and UNOT $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.064 on the test set and 0.062 on the OOD set, against 0.176 and 0.200 for Meta OT, the strongest baseline (Fig. 40).

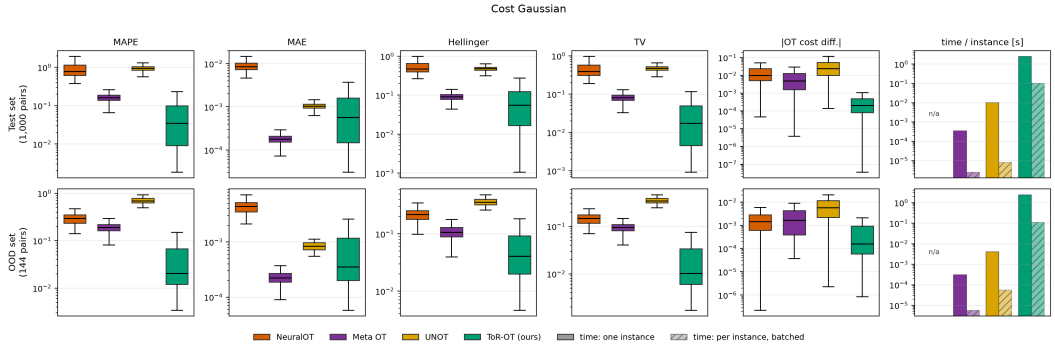

Figure 40: Cost Gaussian: test set (top, 1,000 pairs) and OOD set (bottom, 144 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

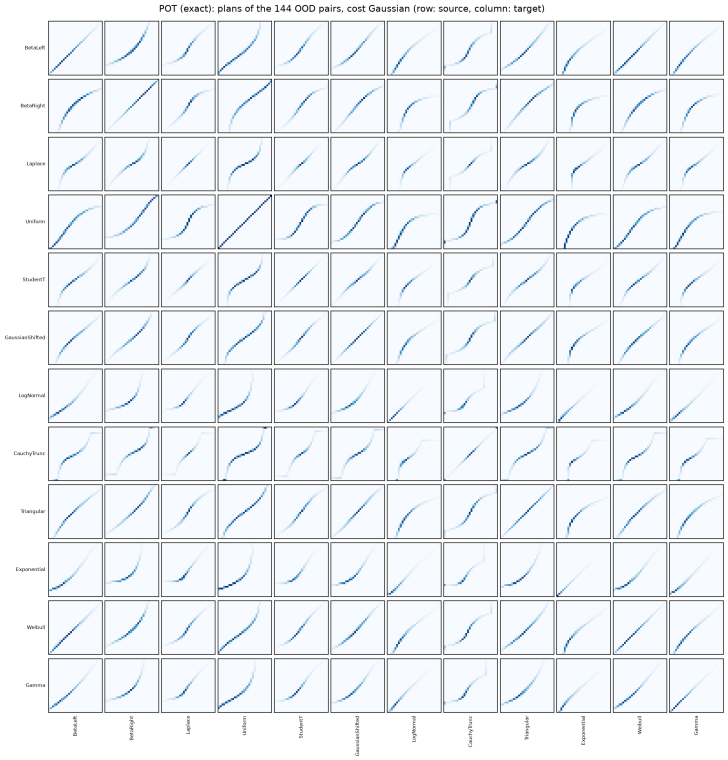

Figure 41: POT (exact): plans of the 144 OOD pairs, cost Gaussian.

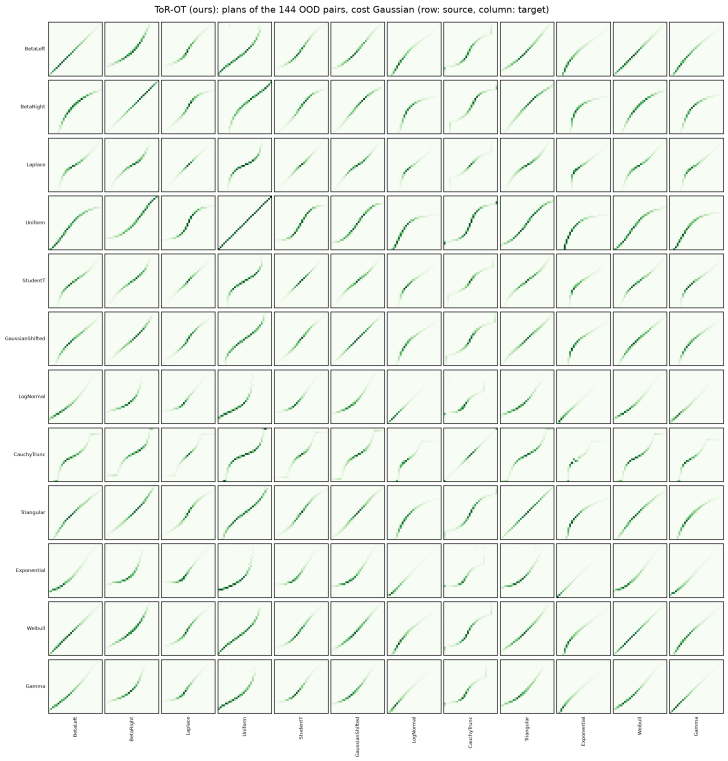

Figure 42: ToR-OT (ours): plans of the 144 OOD pairs, cost Gaussian.

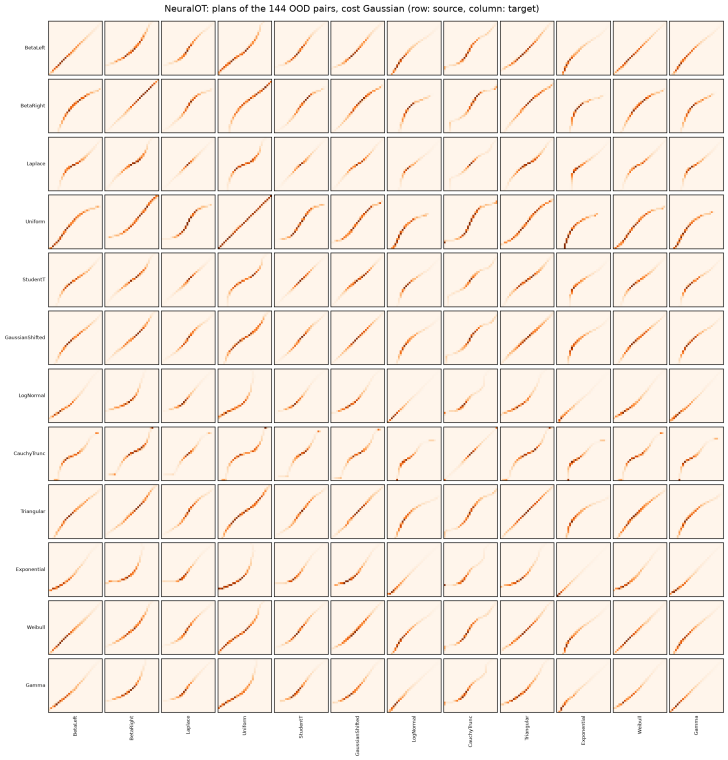

Figure 43: NeuralOT: plans of the 144 OOD pairs, cost Gaussian.

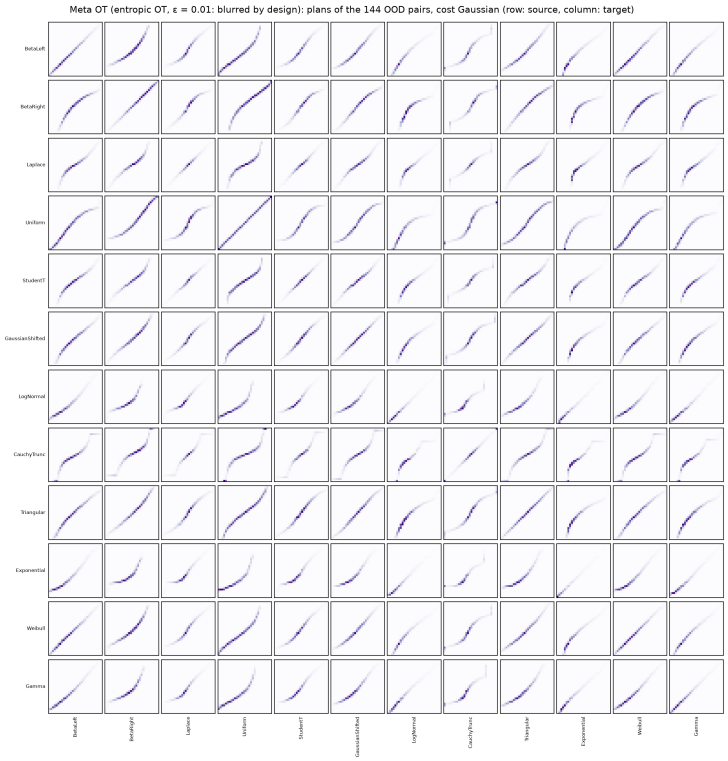

Figure 44: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost Gaussian.

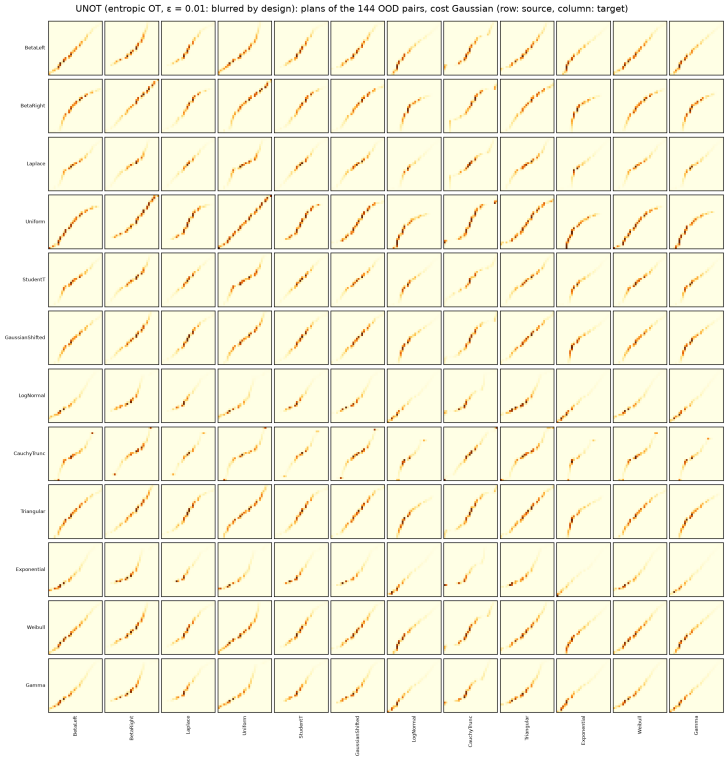

Figure 45: UNOT (entropic, $\varepsilon = 0.01$): plans of the 144 OOD pairs, cost Gaussian.

F.7 All six cost types

Figures 46 and 47 show every method on every cost, one panel per metric. Fig. 48 shows, per OOD pair, how much ToR-OT improves over Meta OT, the strongest baseline on every cost: ToR-OT is better on 129 to 140 of the 144 pairs, with median improvements from 76% to 91%.

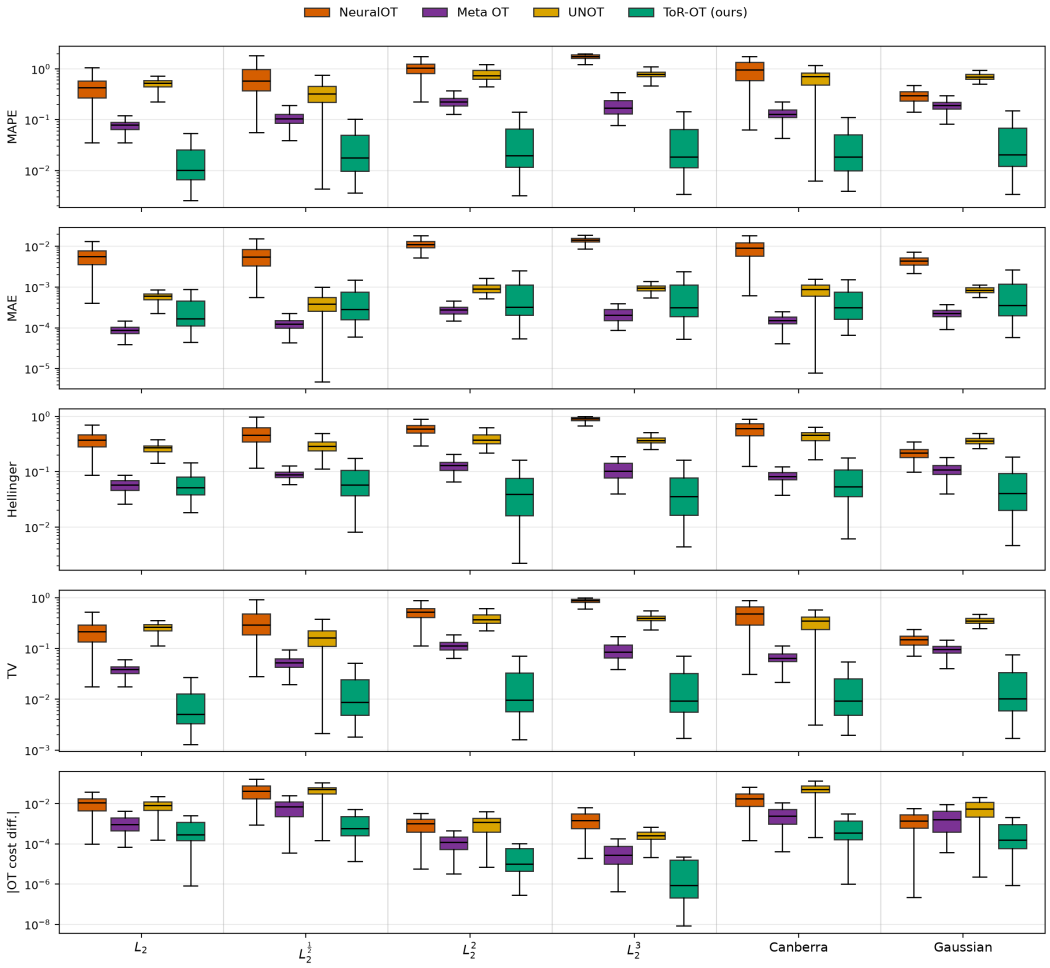

Figure 46: All six costs, OOD set (144 pairs per cost): the five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale).

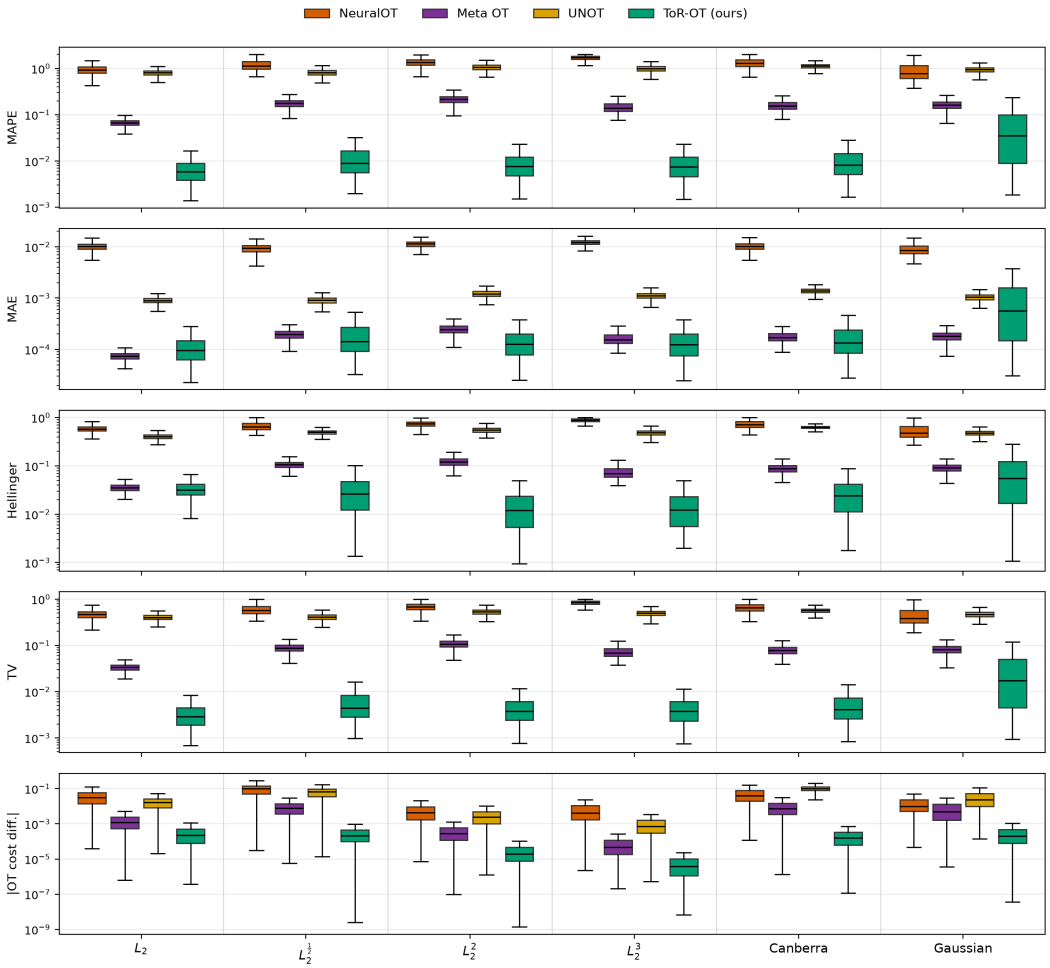

Figure 47: All six costs, test set (1,000 pairs per cost), as Fig. 46.

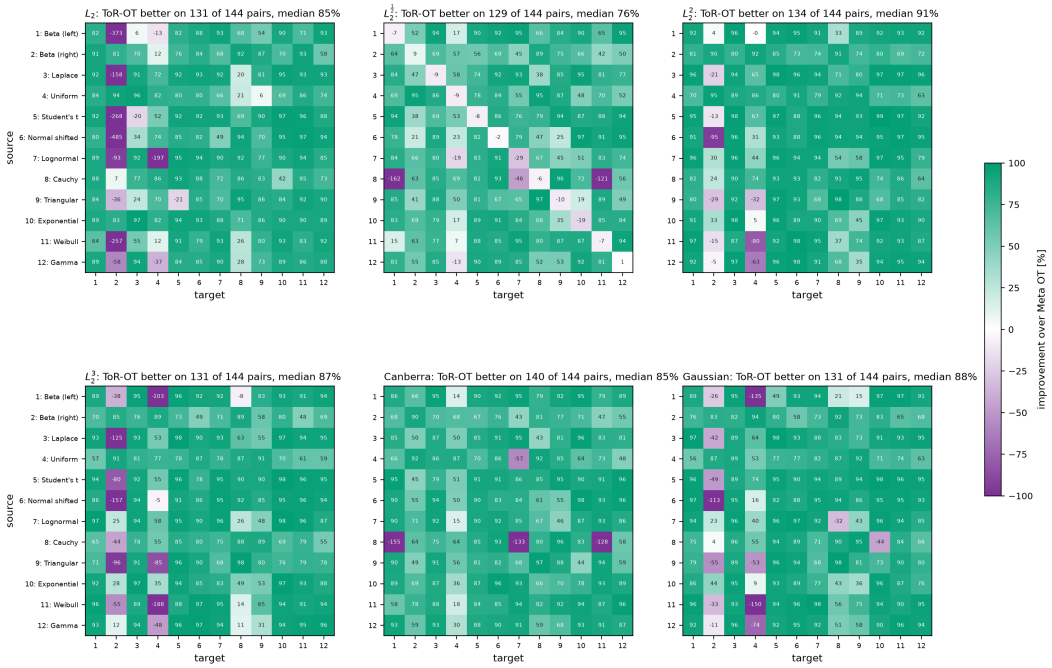

Figure 48: Relative MAPE improvement of ToR-OT over Meta OT, $100\,(\mathrm{MAPE}_{\text{Meta OT}} - \mathrm{MAPE}_{\text{ToR-OT}})/\mathrm{MAPE}_{\text{Meta OT}}$, on each of the 144 OOD pairs (row: source; column: target; distributions numbered as in the first column), for each cost; each method is scored against the solution of its own problem. Green: ToR-OT better; purple: Meta OT better; the color scale is clipped at $\pm 100\%$. Panel titles: the number of pairs on which ToR-OT is better and the median improvement.

G Full results on 2D distributions

Contents

G.1 Cost type: L_2 51

G.2 Cost type: $L_2^{1/2}$ 55

G.3 Cost type: L_2^2 58

G.4 Cost type: L_2^3 61

G.5 Cost type: Canberra 64

G.6 Cost type: Gaussian 67

G.7 All six cost types 70

Each cost-type subsection has a summary of the five plan metrics and the run time (test and OOD sets), followed by the plans of the 36 OOD pairs for POT, ToR-OT, Meta OT and UNOT (and NeuralOT for L_2^2 and L_2).

This appendix repeats App. F for two-dimensional distributions, with the 2D model of App. A. Supports are the centres of a 6×6 grid on $[0, 1]^2$ ($n = m = 36$); the 36×36 plans list the bins with the second coordinate varying fastest. The six costs are those of the 1D model with $\|x - y\|_2$ in place of $|x - y|$ (definitions in each subsection). The two sets:

- **Test set:** 1,000 random pairs per cost, marginals uniform on the simplex (the training distribution).
- **OOD set:** the 36 ordered pairs of 6 named distributions (Fig. 49): the 2D OOD set of the submitted version without 8 Gaussians, which has 24 empty bins of 36, a case the 2D training data never contains (its Dirichlet marginals have no empty bins).

Scoring, metrics and plots are as in App. F: each method is compared with the solution of the problem it solves (exact OT: the optimal plan closest to the prediction; entropic OT: the converged Sinkhorn plan at the method's ε). NeuralOT's 2D implementation supports only L_2^2 and L_2, and it optimizes each pair separately, so it was run on those two costs: all OOD pairs, and 1,000 (L_2^2) and 200 (L_2, spread over the set) test pairs. The box plots use training seed 0 of the learned baselines; the numbers in the text are means over their three seeds.

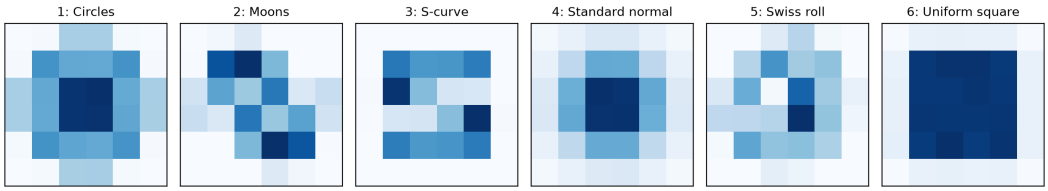

Figure 49: The 6 distributions of the 2D OOD set on the 6×6 grid, numbered as in the plan grids and the improvement heatmaps.

G.1 Cost type: L_2

Cost $c(x, y) = \|x - y\|_2$; Meta OT and UNOT use $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.047 on the test set and 0.036 on the OOD set, against 0.286 and 0.202 for Meta OT, the strongest baseline; ToR-OT is better on 29 of the 36 OOD pairs (Fig. 50). As in 1D, the optimal plan for L_2 is often not unique (POT's plan, Fig. 51, is one of several), and ToR-OT and NeuralOT are scored against the optimal plan closest to their prediction.

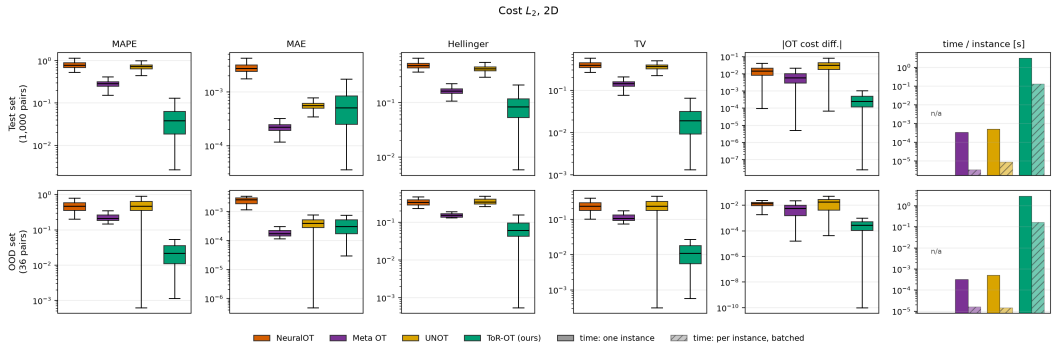

Figure 50: Cost L_2, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a). NeuralOT: 200 of the 1,000 test pairs.

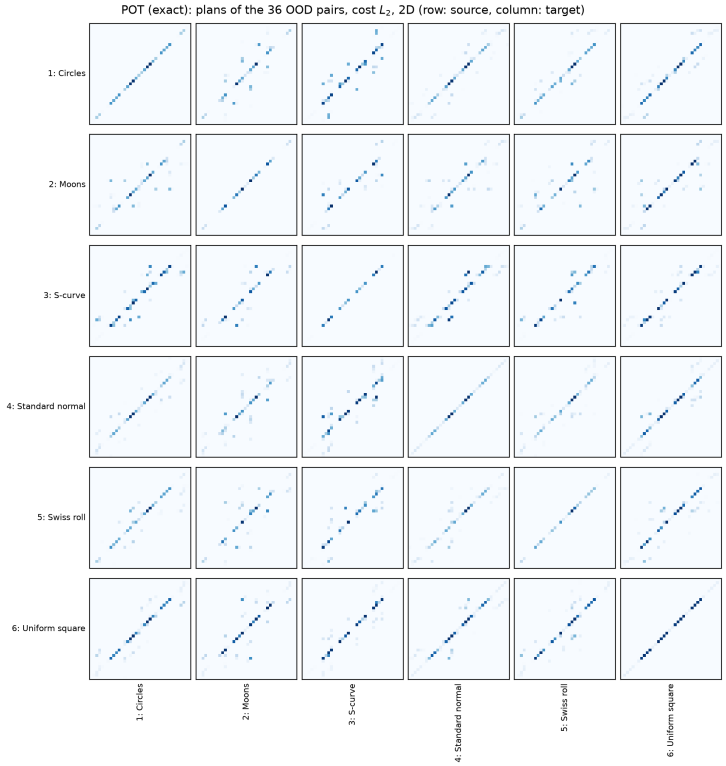

Figure 51: POT (exact): plans of the 36 OOD pairs, cost L_2, 2D.

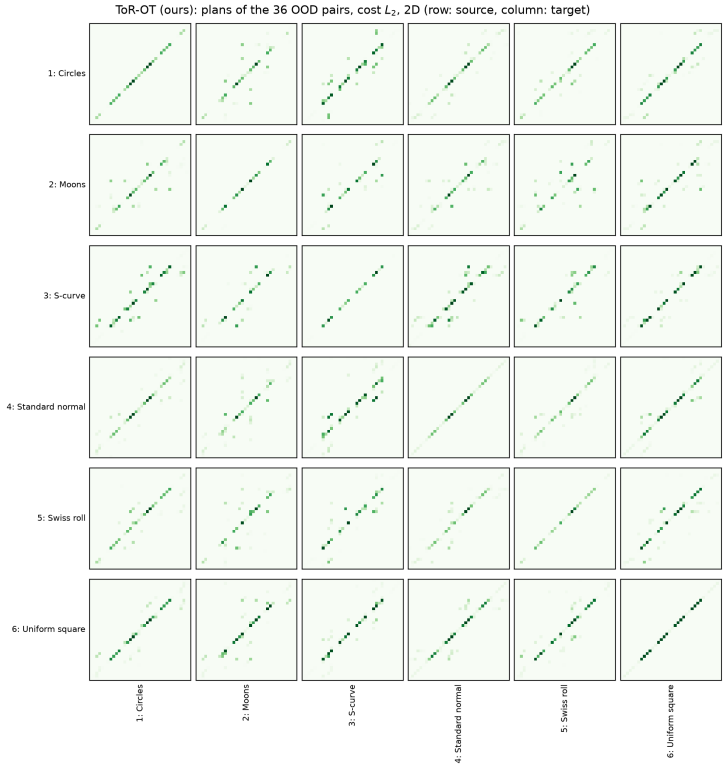

Figure 52: ToR-OT (ours): plans of the 36 OOD pairs, cost L_2, 2D.

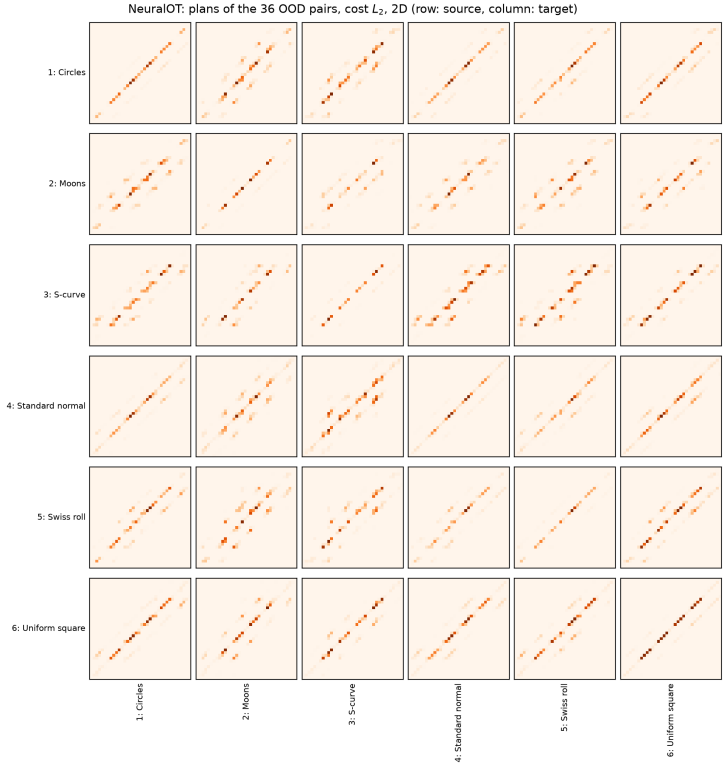

Figure 53: NeuralOT: plans of the 36 OOD pairs, cost L_2, 2D.

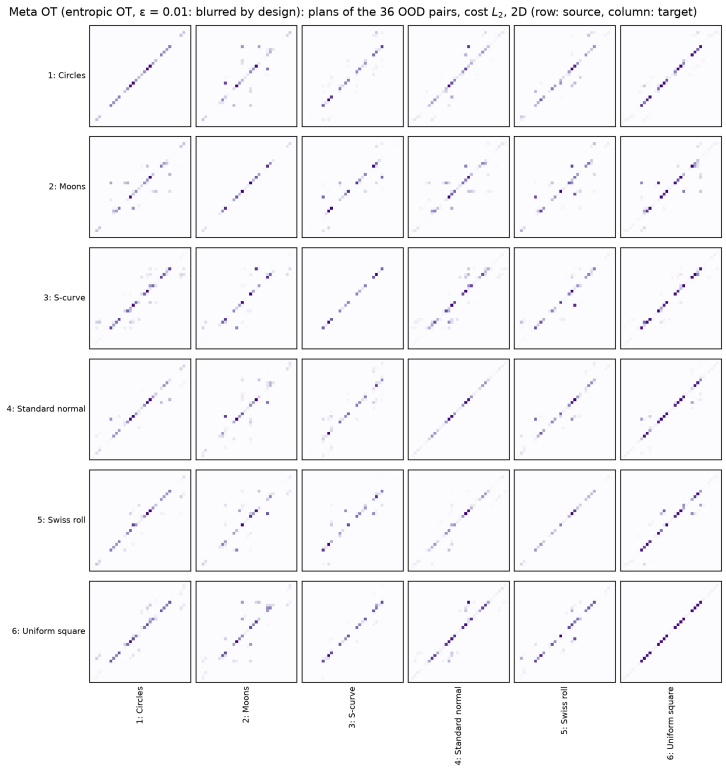

Figure 54: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost L_2, 2D.

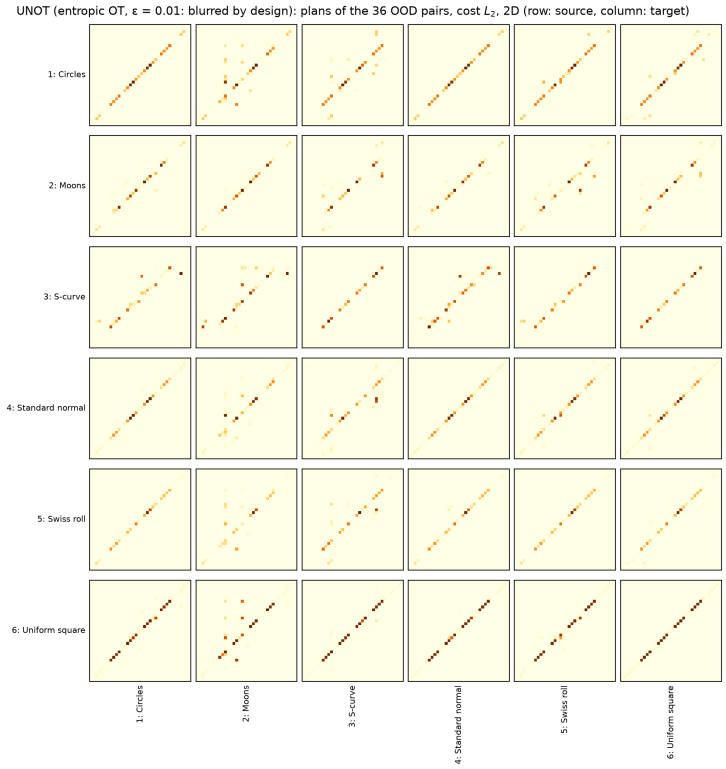

Figure 55: UNOT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost L_2, 2D.

G.2 Cost type: $L_2^{1/2}$

Cost $c(x, y) = \|x - y\|_2^{1/2}$ (concave); Meta OT and UNOT use $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.049 on the test set and 0.032 on the OOD set, against 0.320 and 0.225 for Meta OT, the strongest baseline; ToR-OT is better on 30 of the 36 OOD pairs (Fig. 56).

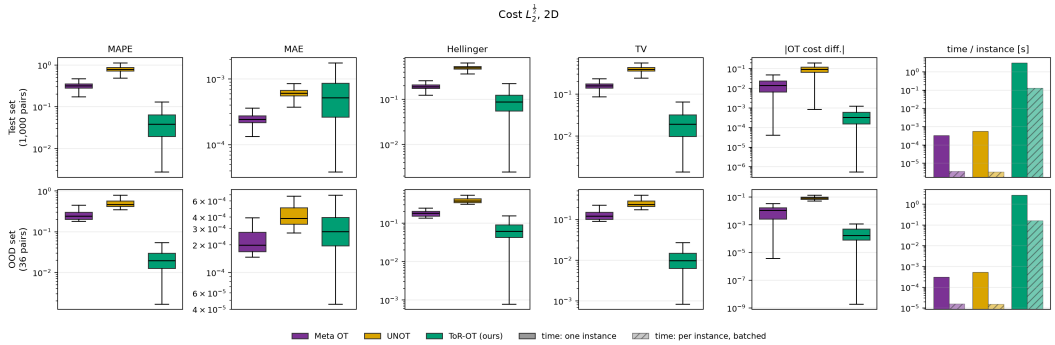

Figure 56: Cost $L_2^{1/2}$, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched).

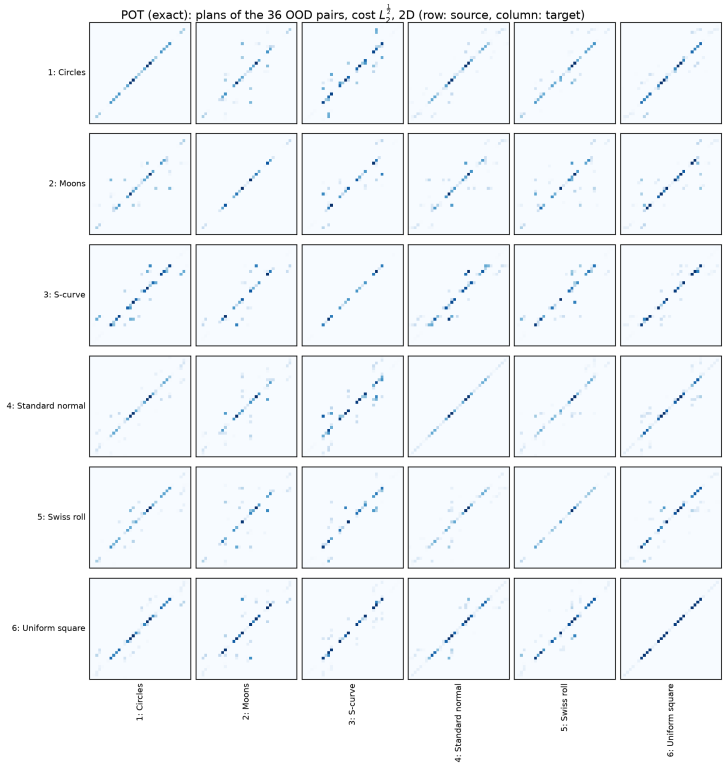

Figure 57: POT (exact): plans of the 36 OOD pairs, cost $L_2^{1/2}$, 2D.

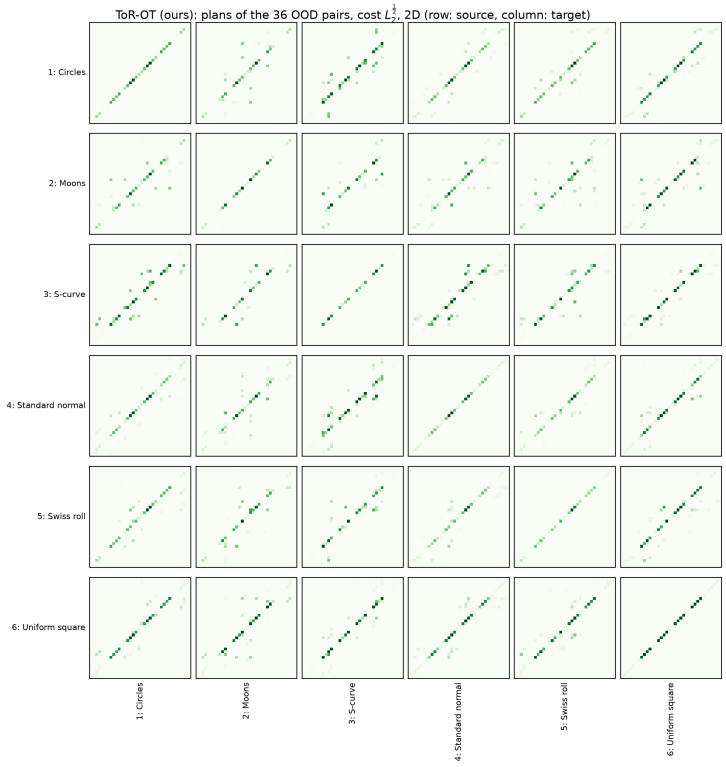

Figure 58: ToR-OT (ours): plans of the 36 OOD pairs, cost $L_2^{1/2}$, 2D.

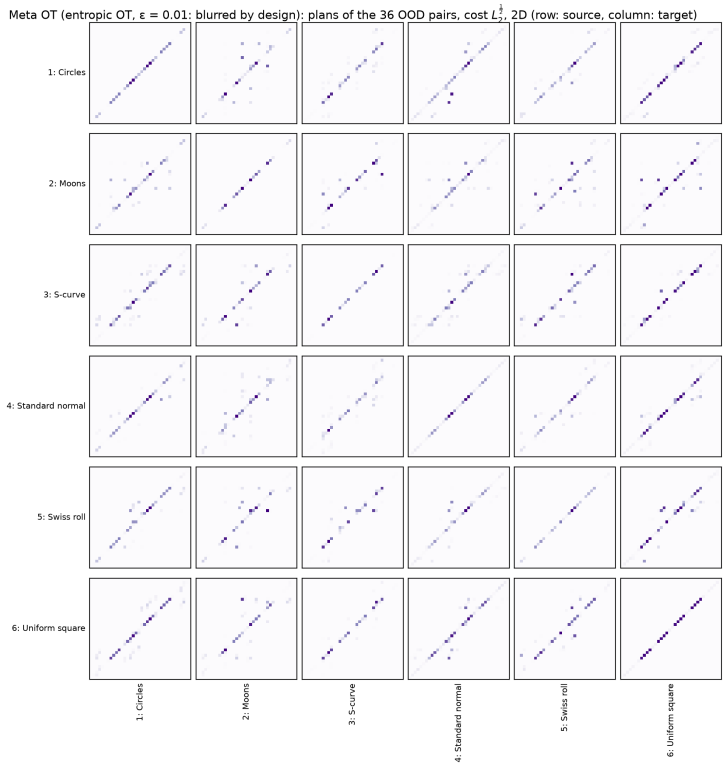

Figure 59: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost $L_2^{1/2}$, 2D.

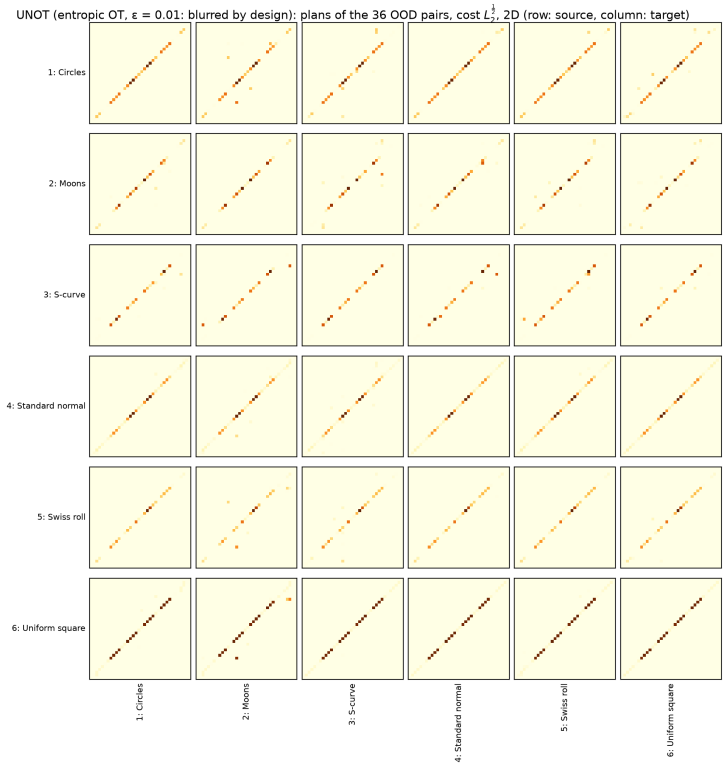

Figure 60: UNOT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost $L_2^{1/2}$, 2D.

G.3 Cost type: L_2^2

Cost $c(x, y) = \|x - y\|_2^2$; Meta OT and UNOT use $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.023 on the test set and 0.029 on the OOD set, against 0.115 and 0.109 for Meta OT, the strongest baseline; ToR-OT is better on 34 of the 36 OOD pairs (Fig. 61).

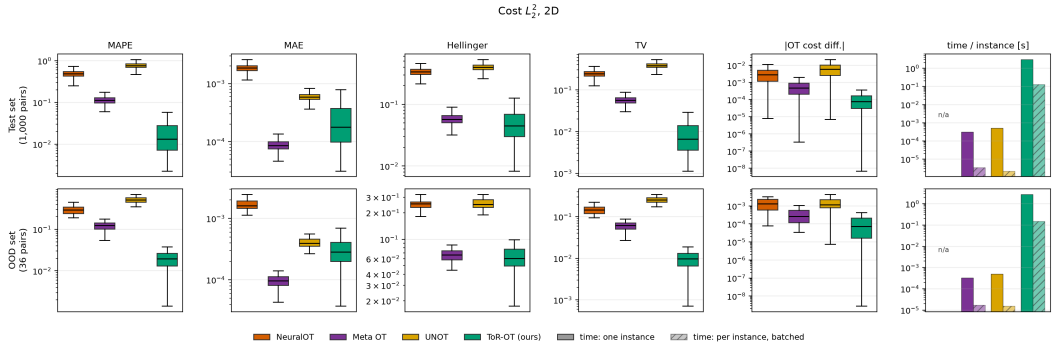

Figure 61: Cost L_2^2, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched). NeuralOT optimizes each instance separately and its runs shared GPUs, so its times are not comparable (n/a).

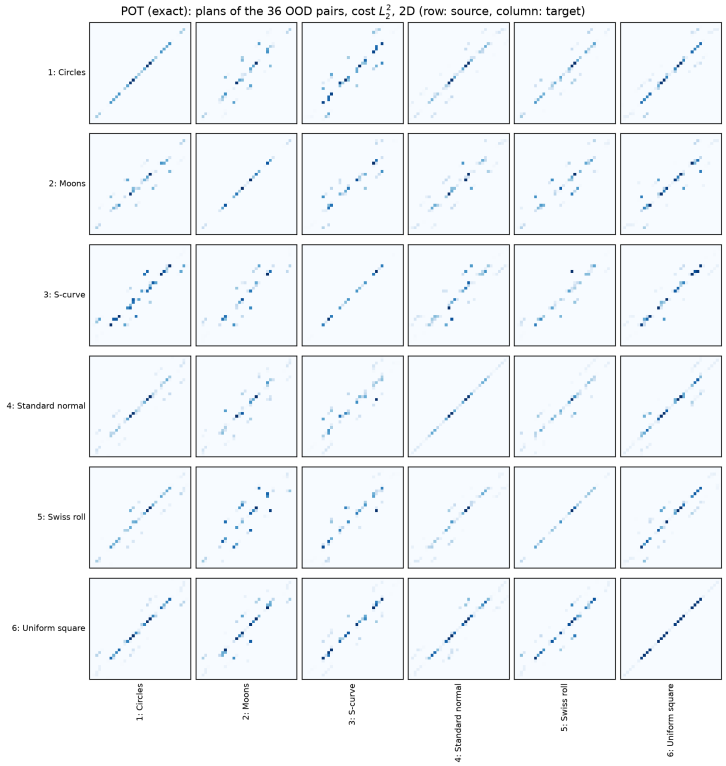

Figure 62: POT (exact): plans of the 36 OOD pairs, cost L_2^2, 2D.

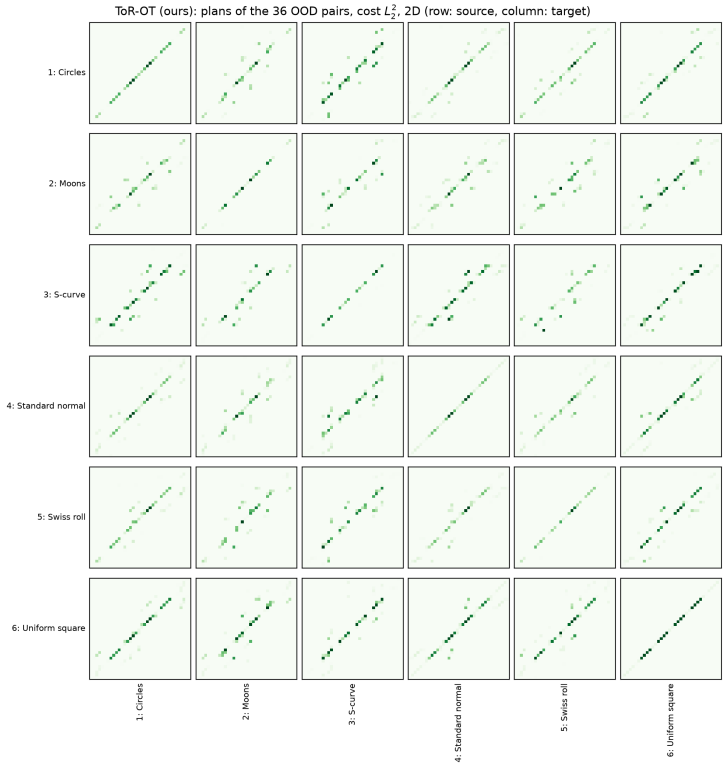

Figure 63: ToR-OT (ours): plans of the 36 OOD pairs, cost L_2^2, 2D.

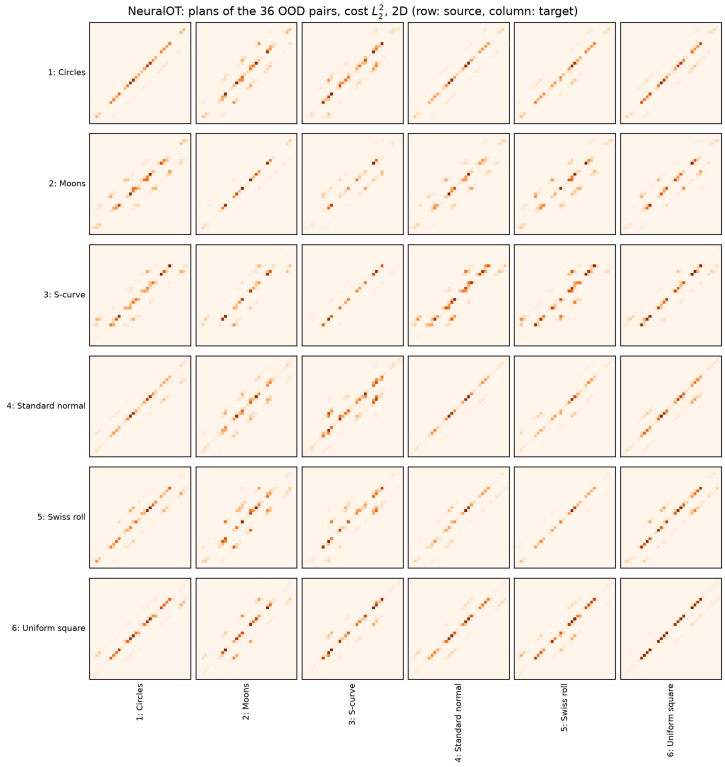

Figure 64: NeuralOT: plans of the 36 OOD pairs, cost L_2^2, 2D.

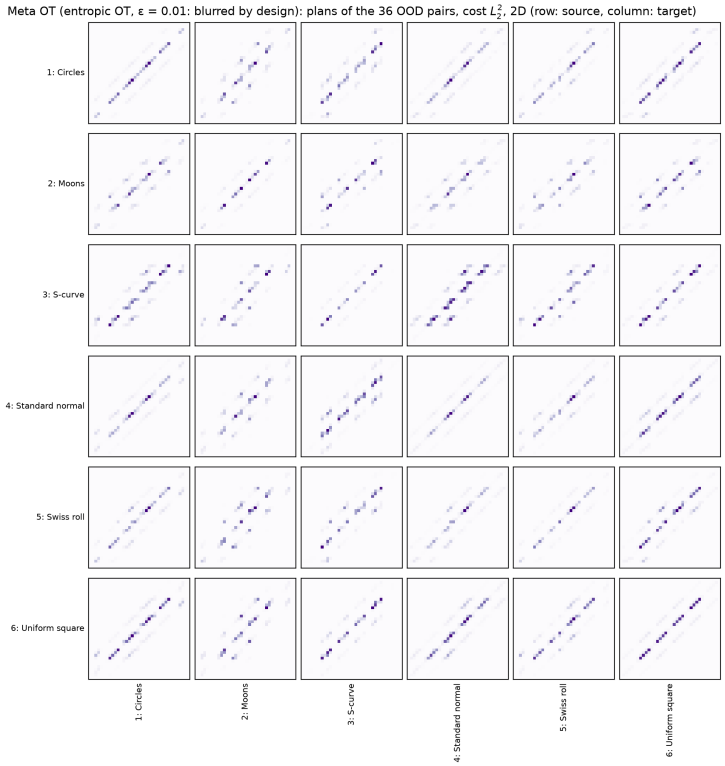

Figure 65: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost L_2^2, 2D.

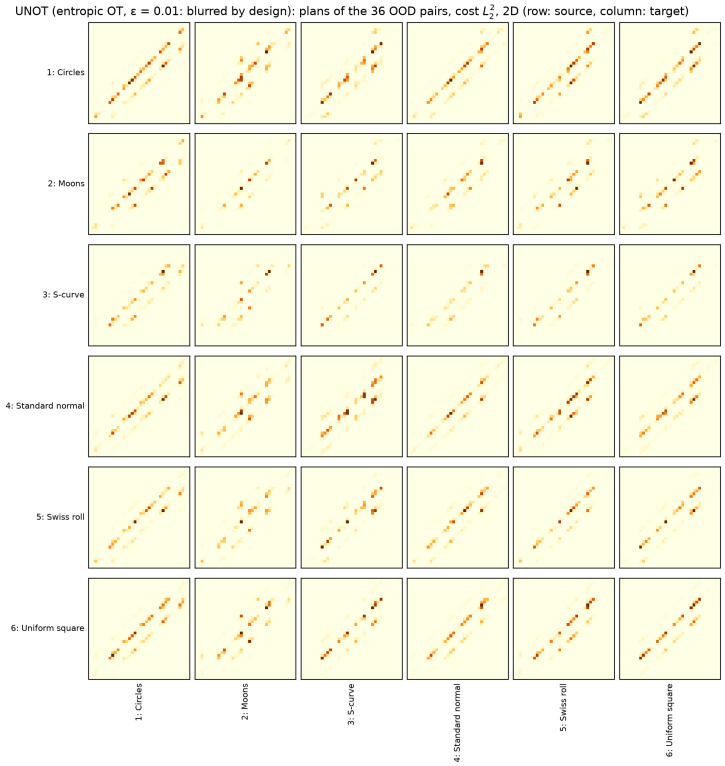

Figure 66: UNOT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost L_2^2, 2D.

G.4 Cost type: L_2^3

Cost $c(x, y) = \|x - y\|_2^3$; Meta OT and UNOT use $\varepsilon = 0.001$. ToR-OT's mean MAPE is 0.080 on the test set and 0.043 on the OOD set, against 0.369 and 0.256 for Meta OT, the strongest baseline; ToR-OT is better on 35 of the 36 OOD pairs (Fig. 67).

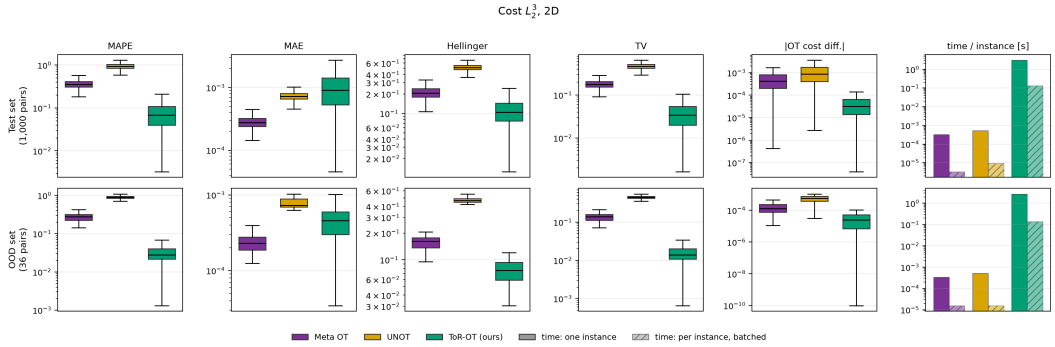

Figure 67: Cost L_2^3, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched).

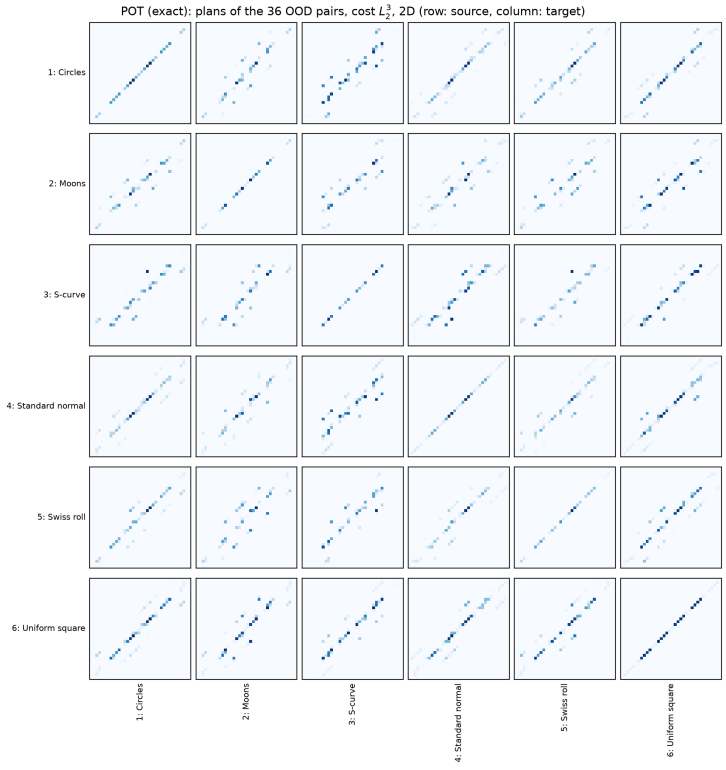

Figure 68: POT (exact): plans of the 36 OOD pairs, cost L_2^3, 2D.

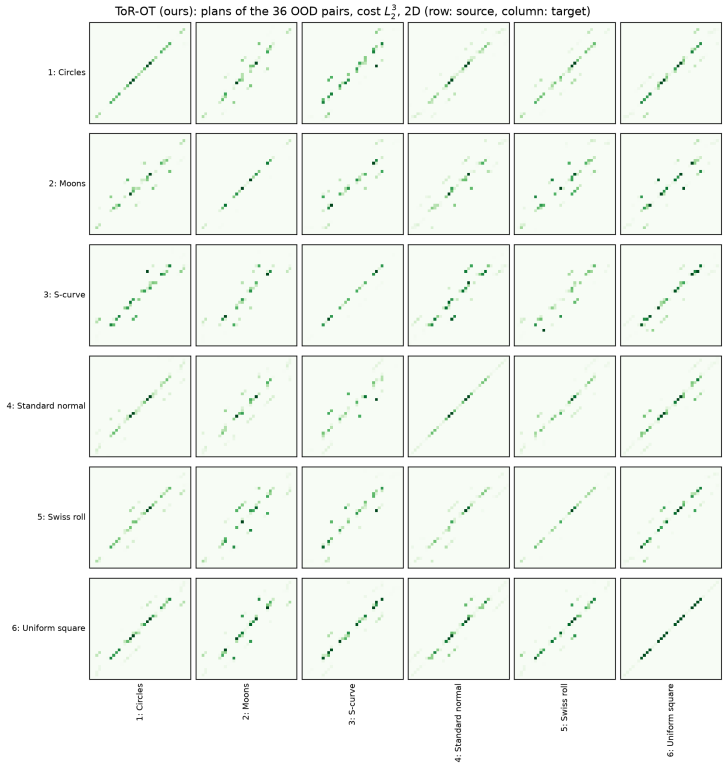

Figure 69: ToR-OT (ours): plans of the 36 OOD pairs, cost L_2^3, 2D.

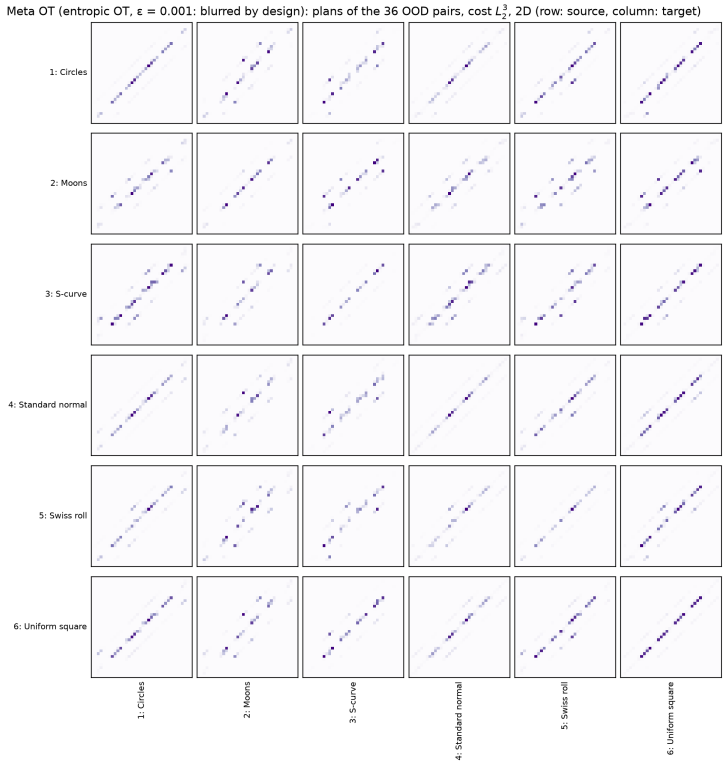

Figure 70: Meta OT (entropic, $\varepsilon = 0.001$): plans of the 36 OOD pairs, cost L_2^3, 2D.

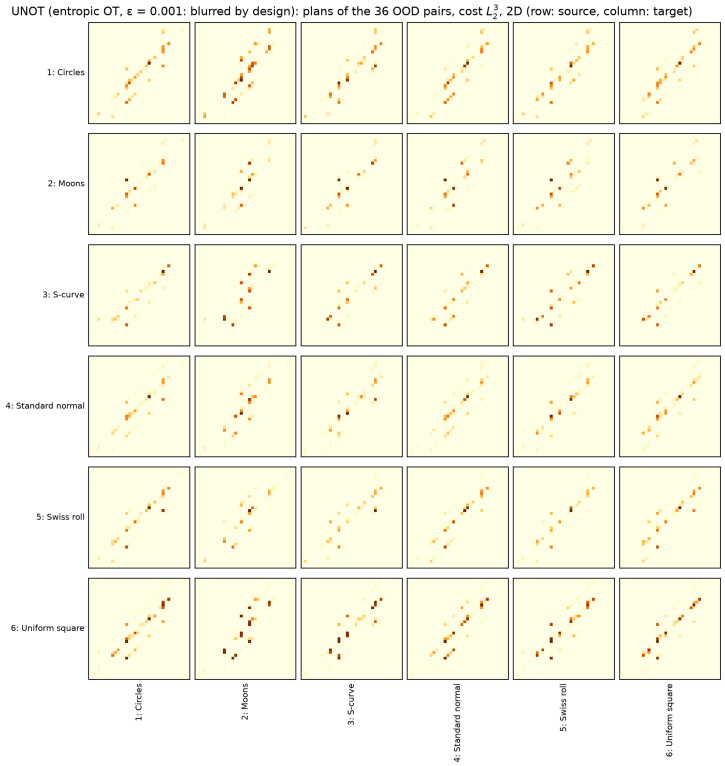

Figure 71: UNOT (entropic, $\varepsilon = 0.001$): plans of the 36 OOD pairs, cost L_2^3, 2D.

G.5 Cost type: Canberra

Cost $c(x, y) = \sum_d |x_d - y_d|/(|x_d| + |y_d|)$; Meta OT and UNOT use $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.081 on the test set and 0.047 on the OOD set, against 0.310 and 0.204 for Meta OT, the strongest baseline; ToR-OT is better on 34 of the 36 OOD pairs (Fig. 72).

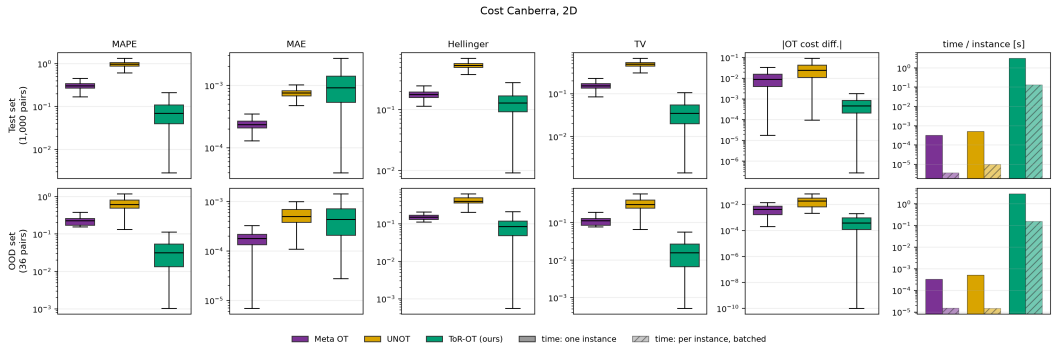

Figure 72: Cost Canberra, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched).

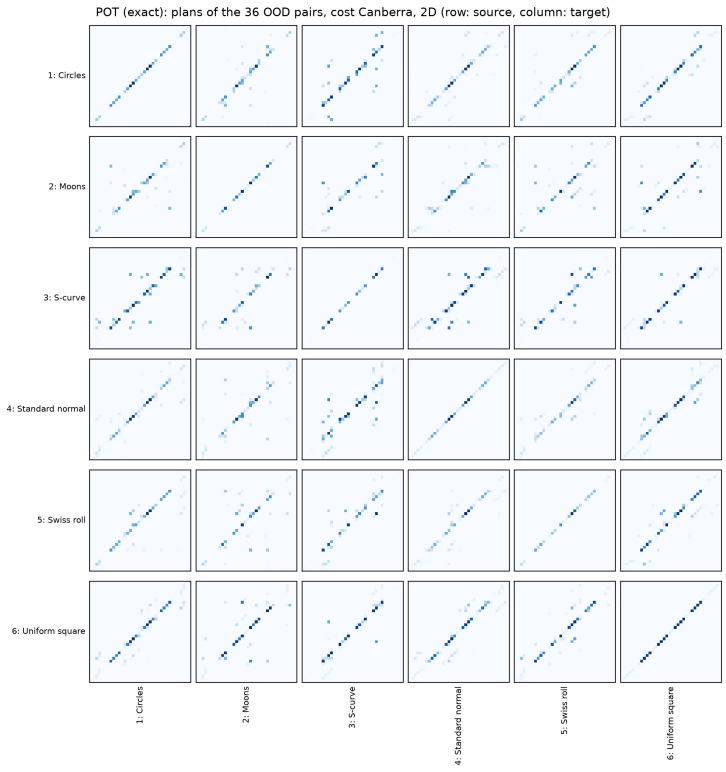

Figure 73: POT (exact): plans of the 36 OOD pairs, cost Canberra, 2D.

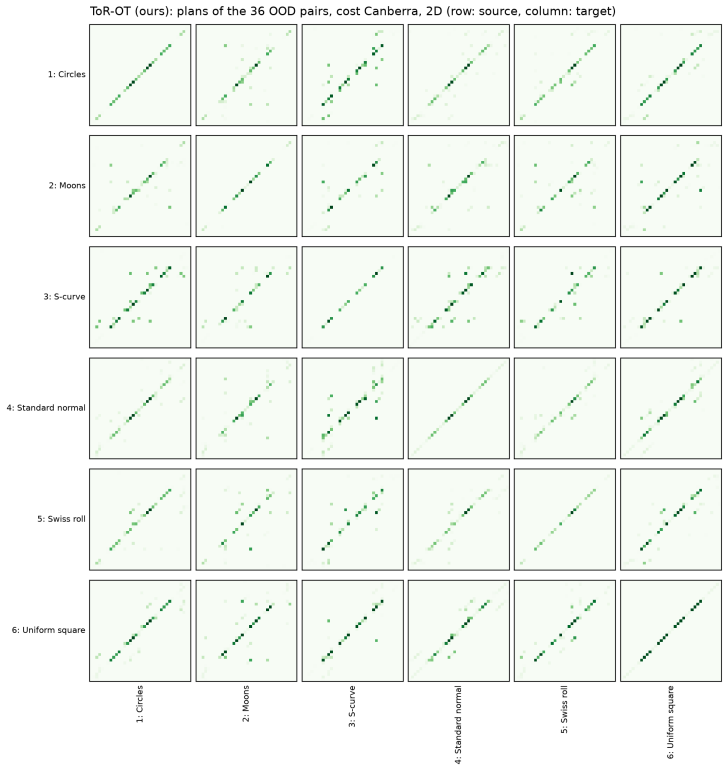

Figure 74: ToR-OT (ours): plans of the 36 OOD pairs, cost Canberra, 2D.

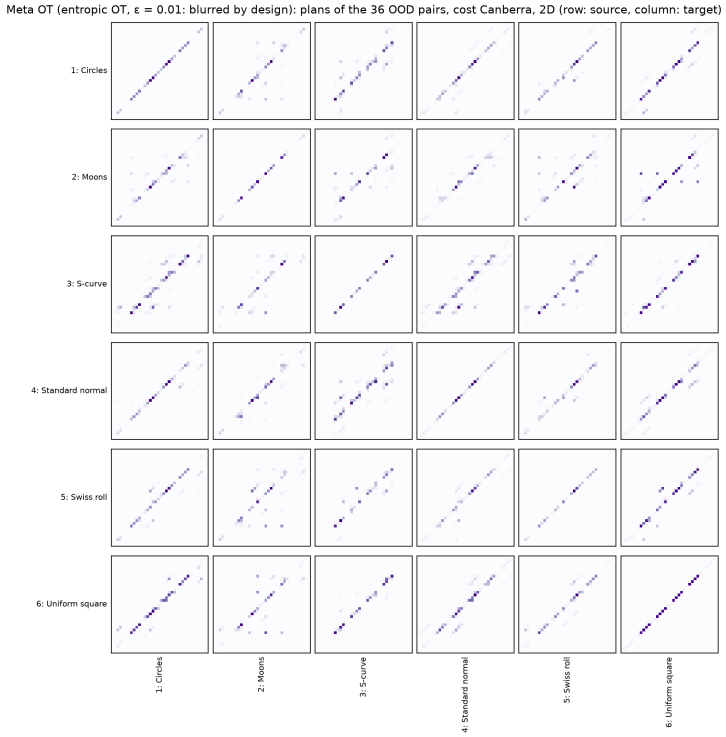

Figure 75: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost Canberra, 2D.

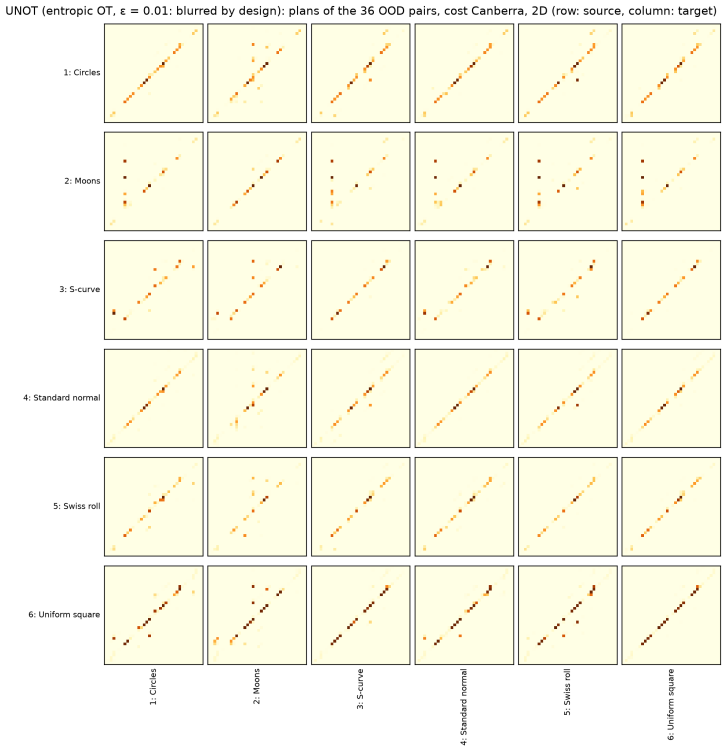

Figure 76: UNOT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost Canberra, 2D.

G.6 Cost type: Gaussian

Cost $c(x, y) = 1 - \exp(-\|x - y\|_2^2/2\sigma^2)$, $\sigma = 0.2$; Meta OT and UNOT use $\varepsilon = 0.01$. ToR-OT's mean MAPE is 0.092 on the test set and 0.055 on the OOD set, against 0.500 and 0.371 for Meta OT, the strongest baseline; ToR-OT is better on 32 of the 36 OOD pairs (Fig. 77).

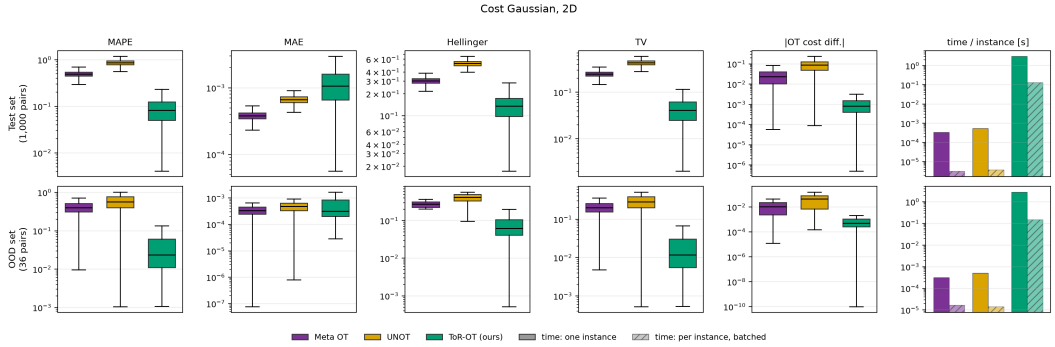

Figure 77: Cost Gaussian, 2D: test set (top, 1,000 pairs) and OOD set (bottom, 36 pairs). Five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale), and time per instance on one A100 GPU (solid: one instance; hatched: per instance, batched).

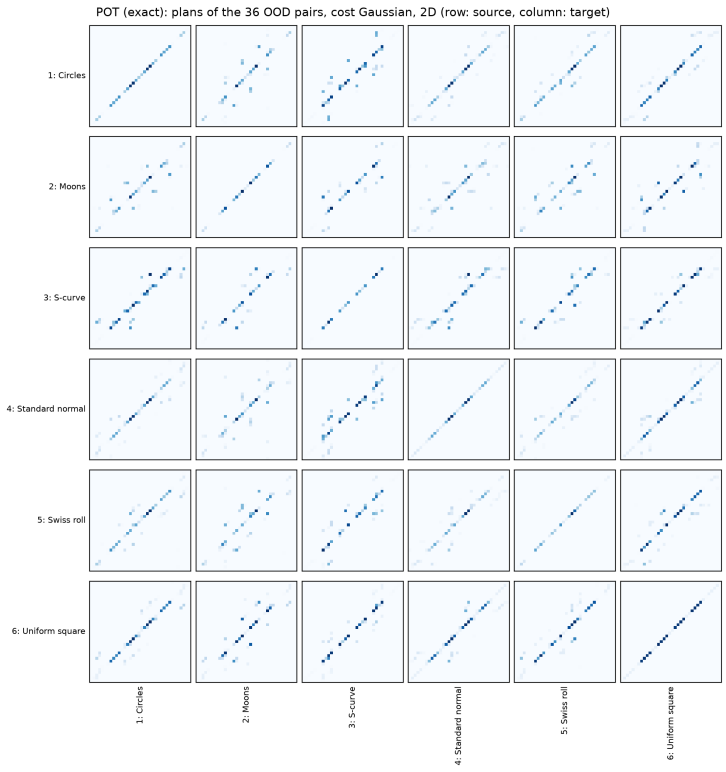

Figure 78: POT (exact): plans of the 36 OOD pairs, cost Gaussian, 2D.

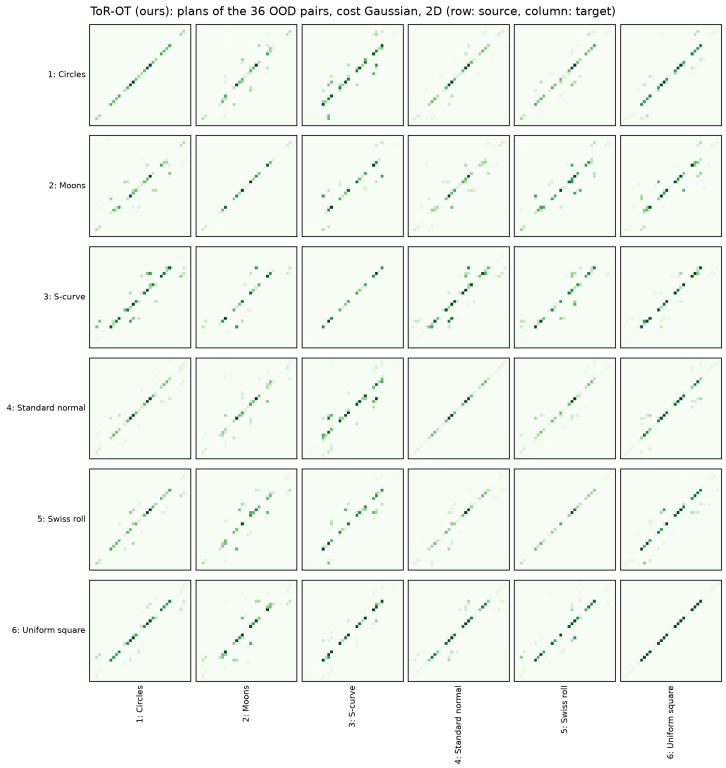

Figure 79: ToR-OT (ours): plans of the 36 OOD pairs, cost Gaussian, 2D.

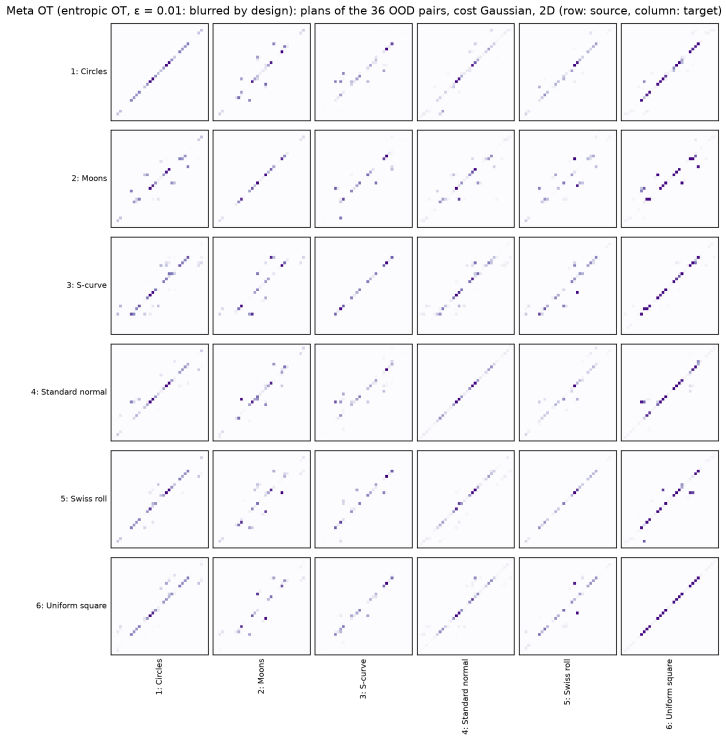

Figure 80: Meta OT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost Gaussian, 2D.

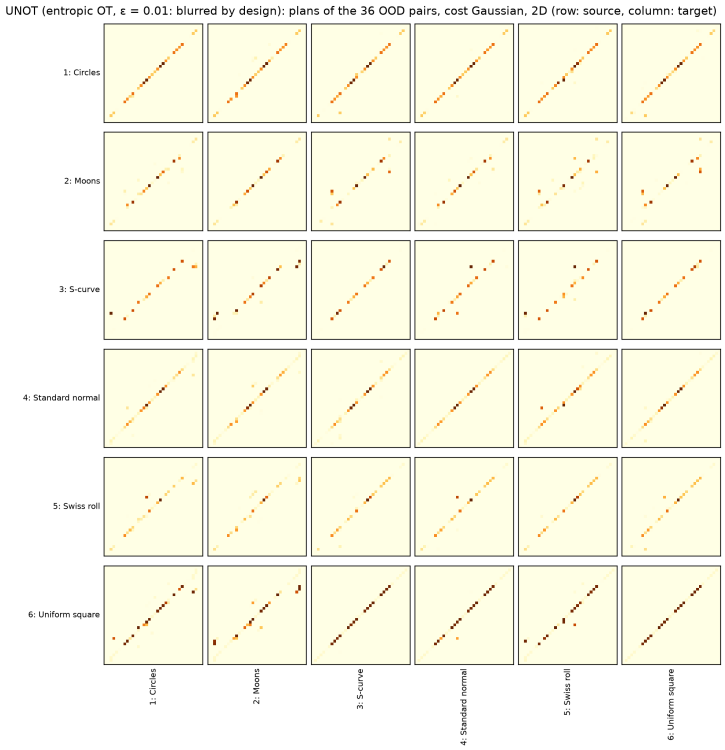

Figure 81: UNOT (entropic, $\varepsilon = 0.01$): plans of the 36 OOD pairs, cost Gaussian, 2D.

G.7 All six cost types

Figures 82 and 83 show every method on every cost, one panel per metric. Fig. 84 shows, per OOD pair, how much ToR-OT improves over Meta OT, the strongest baseline on every cost: ToR-OT is better on 29 to 35 of the 36 pairs, with median improvements from 81% to 90%. Of the 22 pairs (over all six costs) on which Meta OT is better, 18 are on the diagonal (source = target), where both methods are close to the reference (Meta OT's MAPE is lower by at most 0.011).

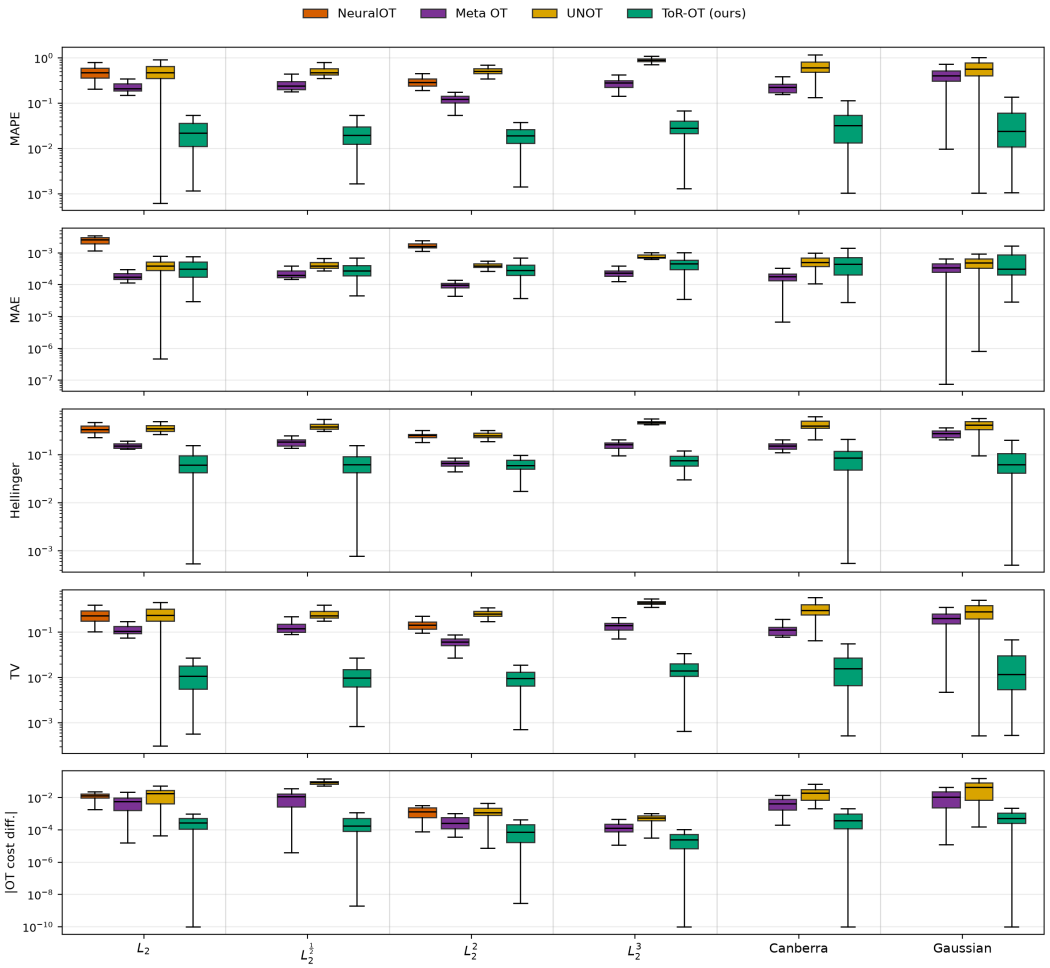

Figure 82: All six costs, 2D OOD set (36 pairs per cost): the five plan metrics, each method against the solution of its own problem (box: quartiles; line: median; whiskers: 1.5 IQR; outliers not shown; log scale). NeuralOT: L_2^2 and L_2 only.

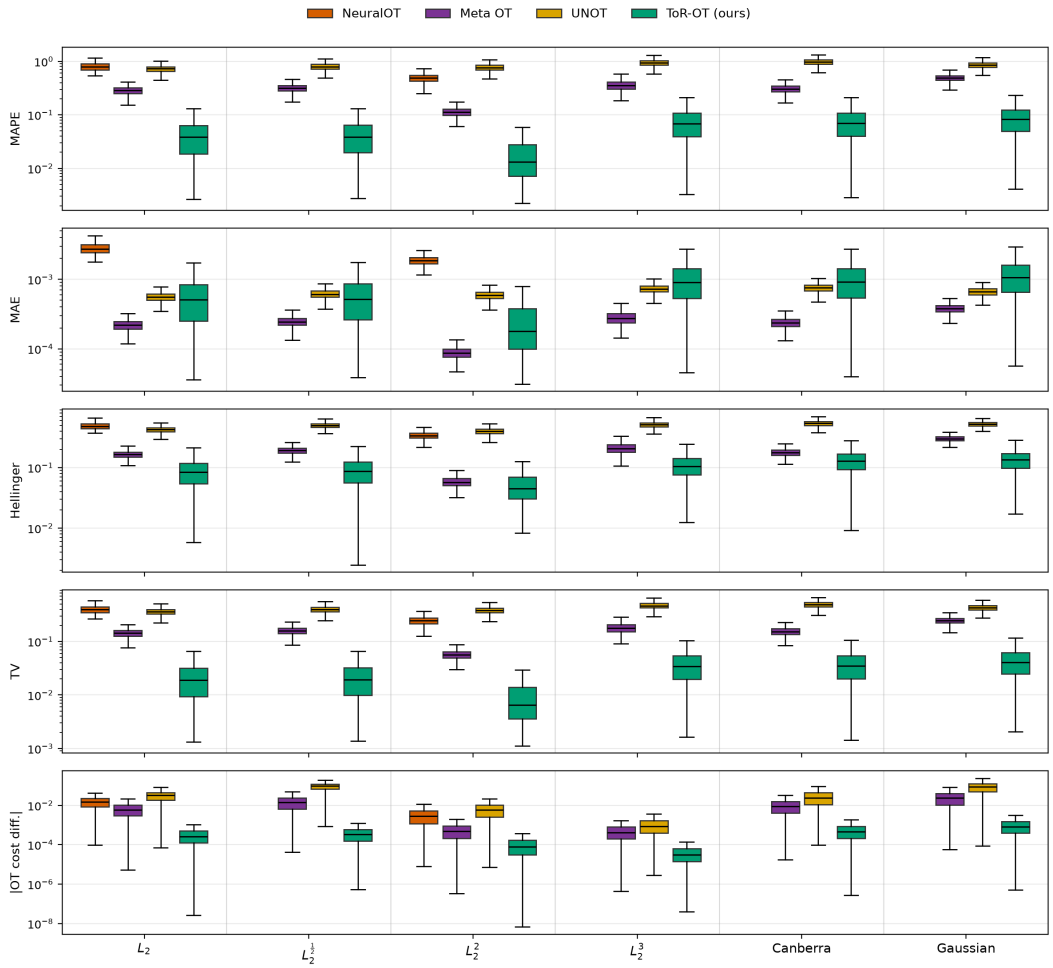

Figure 83: All six costs, 2D test set (1,000 pairs per cost; NeuralOT: 1,000 pairs for L_2^2, 200 for L_2), as Fig. 82.

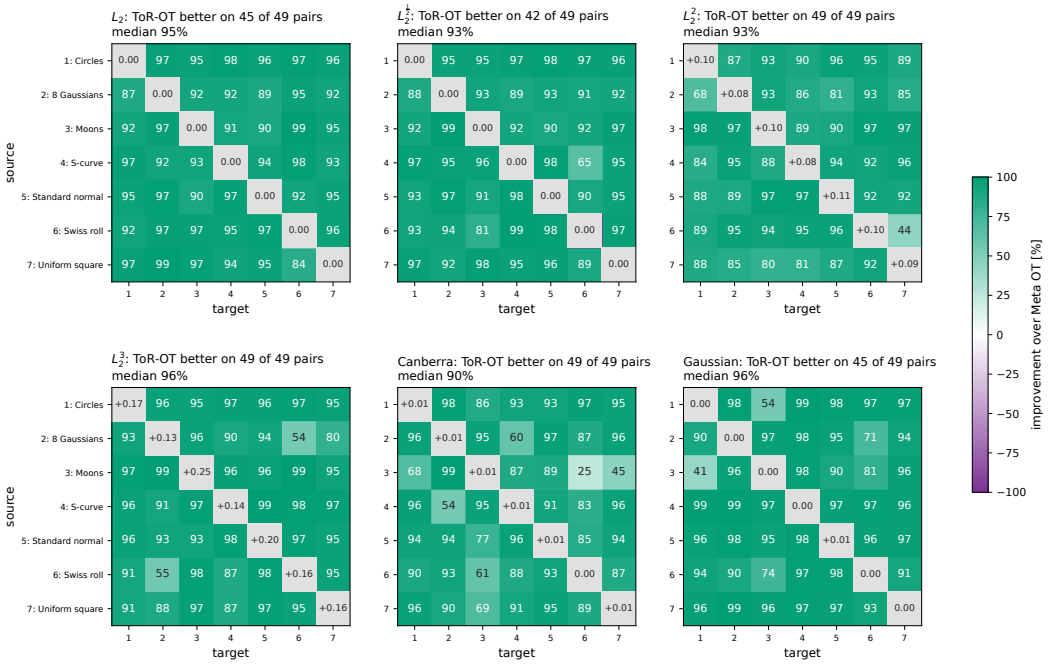

Figure 84: Relative MAPE improvement of ToR-OT over Meta OT, $100\,(\mathrm{MAPE}_{\text{Meta OT}} - \mathrm{MAPE}_{\text{ToR-OT}})/\mathrm{MAPE}_{\text{Meta OT}}$, on each of the 36 2D OOD pairs (row: source; column: target; distributions numbered as in Fig. 49), for each cost; each method is scored against the solution of its own problem. Green: ToR-OT better; purple: Meta OT better; the color scale is clipped at $\pm 100\%$. Diagonal (grey; source = target): the absolute difference $\mathrm{MAPE}_{\text{Meta OT}} - \mathrm{MAPE}_{\text{ToR-OT}}$ instead (positive: ToR-OT better): there both errors can be close to 0 (Meta OT's below 10^{-4} for some costs), so their ratio is not meaningful. Panel titles: the number of pairs on which ToR-OT is better and the median improvement.