TabMT Supplementary Material

A Additional Figures

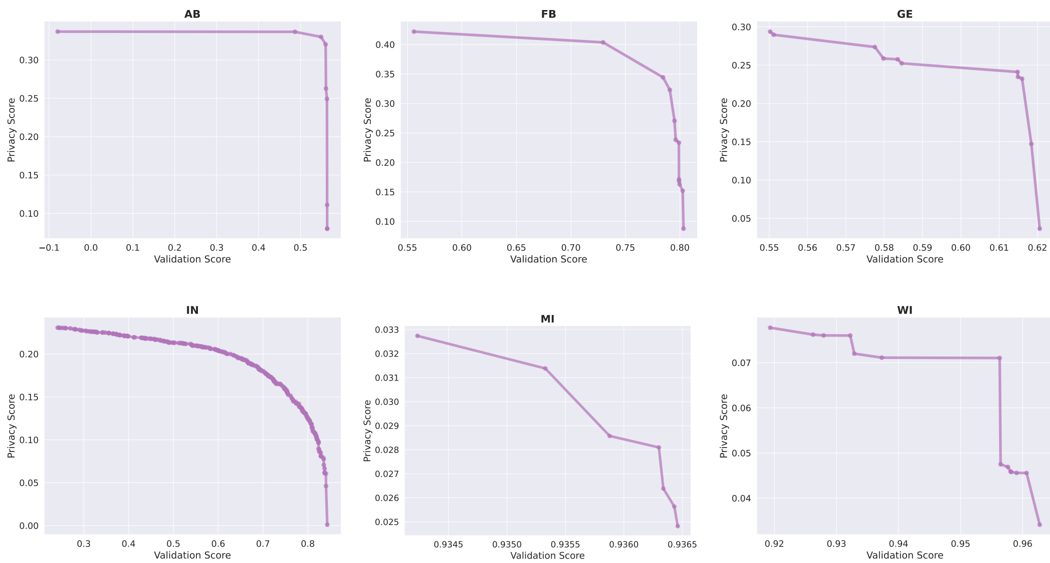

Figure 1: Additional Pareto Fronts

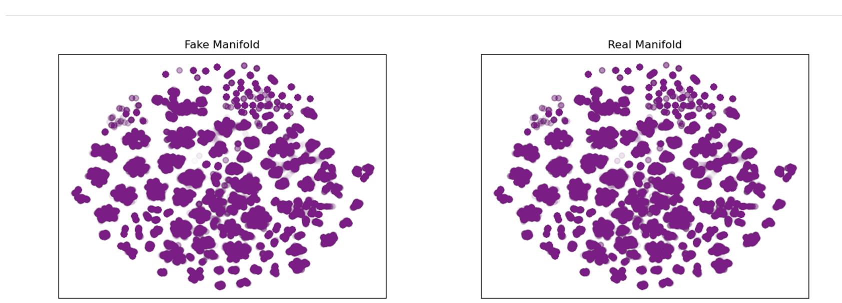

Figure 2: Fake and Real Data Manifolds on the CIDDS-001[10] dataset for TabMT using PaCMAP[12]

B Additional Tables

Hyperparameter	**Search Space**
Network Dim.	Range(16, 512, 16)
Num. of Heads	Choice(1, 2, 4, 8, 16)
Network Depth	Range(2, 12, 2)
Drop Path Rate	Range(0.0, 0.25, 0.05)
Dropout Rate	Range(0.0, 0.5, 0.05)
Learning Rate	LogInterval(5e-5, 5e-3)
Weight Decay	LogInterval(0.001, 0.03)
Max Steps	Range(10000, 40000, 5000)
Batch Size	Range(256, 2048, 256)
Max Bins	Range(10, 200, 10)
Data Multiplier	8
Gen. Batch Size	512
Optimizer	AdamW[7]
Annealing	Cosine
Quantizer	KMeans
Num. Trials	50

Table 1: The hyperparameter search space for training TabMT. Max bins represents the maximum number of bins used during quantization of continuous attributes. We find many hyperparameters in this search space to be unimportant to final model quality, as such the space can likely be pruned for better tuning results. We use the Optuna[1] framework with default settings for optimization

Field	**Type**
Date	timestamp
Source IP	categorical
Destination IP	categorical
Protocol	categorical
Source Port	categorical
Destination Port	categorical
Duration	continuous
Bytes	ordinal
Packets	ordinal
Flags	categorical
Type of Service	categorical

Table 2: Original Netflow Structure

Field	**Cardinality**
Weekday	7
Hour	24
Minute	60
Second	60
Millisecond	1000
Source IP	51092
Destination IP	50923
Protocol	5
Source Port	61700
Destination Port	61448
Duration	37787
Bytes	181536
Packets	10489
Flags	34
Type of Service	5

Table 3: Post-processed Netflow Structure. All fields are processed to be categorical variables for training.

Hyperparameter	**Search Space**
τ_i	Interval(0.5, 5.0)
sampler	NSGAII[3]

Table 4: Hyperparamters used when tuning temperatures, we search a separate temperature for each field, optimizing privacy and ML Efficiency simultaneously. This can certainly be done more efficiently, as NSGAII is sample inefficient for this use case. We also use the full 8x data multiplier when searching here, using a smaller multiplier or number of trial repeats during search would speed this up significantly. We didn't use an explicit trial budget and qualitatively evaluated convergence. This took 0.25-5 GPU days except for MI and FB which each took roughly 15.

DS	CTabGAN+	**TabMT**
AB	0.075(0.467)	**0.249**(0.533)
AD	0.119(0.772)	**1.01**(0.811)
BU	0.164(0.864)	**0.165**(0.908)
CA	0.056(0.525)	**0.117**(0.832)
CAR	0.012(0.733)	**0.041**(0.737)
CH	0.212(0.702)	**0.281**(0.758)
DI	0.196(0.734)	**0.243**(0.740)
FB	0.427(0.509)	**0.429**(0.566)

Table 5: DCR scores for CTabGAN+ and TabMT. MLE scores are in parentheses. We win on both privacy and MLE for all tested datasets. Our increased privacy on FB vs. the main privacy table demonstrates our models unique ability to control the trade-off between MLE and DCR

DS	STaSY	**TabMT**
AB	0.482	**0.535**
AD	0.790	**0.814**
BU	0.881	**0.908**
CA	0.762	**0.838**
CAR	0.725	**0.738**
CH	0.738	**0.741**
DI	0.727	**0.769**

Table 6: Preliminary MLE scores for a recent work, STaSY[4], compared to TabMT. TabMT obtains a higher score on all tested datasets. MLP Width, depth, and learning rate were included in the search.

C Pseudocode

Snippet 1: Ordered Embedding

```
class OrderedEmbedding:
    def __init__(self, occ: Tensor, dim: int):
        # occ: ordered cluster centers
        self.E = zeros(len(occ), dim)
        self.l = randn(dim) * 0.05
        self.h = randn(dim) * 0.05
        self.r = (occ - occ[0]) / (occ[-1] - occ[0])

    @property
    def weight(self):
        return self.r * self.l + (1 - self.r) * self.h + self.E

    def forward(self, idx: Tensor):
        return self.weight[idx]
```

Snippet 2: Dynamic Linear layer

```
class DynamicLinear:
    def __init__(self, embedding: OrderedEmbedding | Embedding):
        self.E = embedding
        self.temp = ones(1)
        self.bias = zeros(len(embedding))

    def forward(self, x: Tensor):
        raw_logits = x @ self.E.weight.T + self.bias
        return raw_logits / sigmoid(self.temp)
```

Snippet 3: Training step

```
def training_step(mdl: MaskedTransformer, batch: Tensor):
    mask = rand_like(batch) > rand(len(batch), 1)
    preds = mdl(batch, mask)
    batch[~mask] = -1 #ignore unmasked entries in loss
    loss = cross_entropy(preds, batch)

    return loss
```

Snippet 4: Batched generation

```
def gen_batch(mdl: MaskedTransformer, n: int, l: int, temps: Tensor):
    batch = zeros(n, l)
    mask = ones_like(batch)

    for i in randperm(l):
        preds = mdl(batch, mask)
        batch[:, i] = Categorical(logits=preds[i] / temps[i]).sample()
        mask[:, i] = False
    return batch
```

`MaskedTransformer`: We use a standard Transformer Encoder architecture. We use learned positional embeddings initialized with normal distribution with $\sigma = 0.01$. We use a LayerNorm before the DynamicLinear layers. Our mask token is intialized with a normal distribution as well with $\sigma = 0.05$, to match the other embeddings.

`Embedding`: We use a standard embedding initialized with a normal distribution and $\sigma = 0.05$, and no magnitude clipping or rescaling during training.

D Dataset Info

For comparison, we use the same datasets and splits as Kotelnikov et al. [6] in our main comparison. We list them below along with their sources.

- **AB**: Abalone: OpenML, CC Licensce
- **AD**: Adult ROC[5]: UCI Data Repo, CC License
- **BU**: Adopt a Buddy: Kaggle, Other
- **CA**: California Housing[9]: Kaggle, CC License
- **CAR**: Cardiovascular Disease: Kaggle, CC License
- **CH**: Churn Modeling: Kaggle, CC License
- **DI**: Diabetes: OpenML, CC Licensce
- **FB**: Facebook Comment Volume[11]: UCI Data Repo, CC License
- **GE**: Gesture Phase Prediction[8]: UCI Data Repo, CC License
- **HI**: Higgs (98K)[2]: OpenML, CC License
- **HO**: House 16H: OpenML, CC Licensce
- **IN**: Medical Costs, Kaggle, ODbL
- **KI**: King Count Housing Prices: OpenML, CC License
- **MI**: MiniBOONE Particle Prediction: OpenML, CC License
- **WI**: Wilt Remote Sensing: OpenML, CC License

We additionally use the CIDDS-001[10] dataset for scaling experiments, which has a CC license.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):4308, 2014.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

[4] Jayoung Kim, Chaejeong Lee, and Noseong Park. Stasy: Score-based tabular data synthesis. *arXiv preprint arXiv:2210.04018*, 2022.

[5] Ron Kohavi et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207, 1996.

[6] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. *arXiv preprint arXiv:2209.15421*, 2022.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[8] Renata Cristina Barros Madeo, Sarajane Marques Peres, and Clodoaldo Aparecido de Moraes Lima. Gesture phase segmentation using support vector machines. *Expert Systems with Applications*, 56:100–115, 2016.

[9] R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[10] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European conference on cyber warfare and security. ACPI*, pages 361–369, 2017.

[11] Kamaljot Singh. Facebook comment volume prediction. *International Journal of Simulation: Systems, Science and Technologies*, 16(5):16–1, 2015.

[12] Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021. URL `http://jmlr.org/papers/v22/20-1061.html`.