A supplemental material

A.1 Details of Meta-sketch Operations

Algorithm 3 describes details about operations of the meta-sketch, including broadcast, dimensions conversion processes, and three forms of $\ominus$. It is important to emphasize that all the operations are performed on one stream item, so that a parallelized version can easily be implemented by adding an additional dimension.

Algorithm 3: Details of Meta-sketch Operations

Operation `Store`(e_i, M)**:**
- $z_i, r_i \leftarrow \mathcal{F}_E(e_i)$; $a_i \leftarrow \mathcal{F}_{Sa}(r_i)$;
- $a_i \leftarrow changeShape(a_i, \mathbb{R}^{d_1 \times d_2}, \mathbb{R}^{d_1 \times 1 \times d_2})$
- $z_i \leftarrow changeShape(z_i, \mathbb{R}^{l_z}, \mathbb{R}^{d_1 \times l_z \times 1})$
- $M \leftarrow M + z_i a_i$;

Operation `Delete`(e_i, M)**:**
- $z_i, r_i \leftarrow \mathcal{F}_E(e_i)$; $a_i \leftarrow \mathcal{F}_{Sa}(r_i)$;
- $a_i \leftarrow changeShape(a_i, \mathbb{R}^{d_1 \times d_2}, \mathbb{R}^{d_1 \times 1 \times d_2})$
- $z_i \leftarrow changeShape(z_i, \mathbb{R}^{l_z}, \mathbb{R}^{d_1 \times l_z \times 1})$
- $M \leftarrow M - z_i a_i$;

Operation `Query`(x_i, M, N)**:**
- $z_i, r_i \leftarrow \mathcal{F}_E(x_i)$; $a_i \leftarrow \mathcal{F}_{Sa}(r_i)$;
- $\hat{f}_i \leftarrow \mathcal{F}_{dec}(\{M \ominus a_i\}, z_i, N)$;
- **return** $\hat{f}_i$;

Module `Embeding`(x_i)**:**
- $z_i \leftarrow g_{emb}(e_i)$; $r_i \leftarrow g_{add}(z_i)$;
- **return** z_i, r_i

Module `SparseAddress`(r_i)**:**
- $r_i \leftarrow changeShape(r_i, \mathbb{R}^{l_r}, \mathbb{R}^{d_1 \times 1 \times l_r})$;
- $\hat{a}_i \leftarrow r_i A$;
- $\hat{a}_i \leftarrow changeShape(\hat{a}_i, \mathbb{R}^{d_1 \times 1 \times d2}, \mathbb{R}^{d_1 \times d_2})$;
- $a_i \leftarrow SparseMax(\hat{a}_i, dim = -1)$
- **return** a_i

Module `Decoding`$(\{M \ominus a_i\}, z_i, N)$**:**
- $m_i \leftarrow basicRead(M, a_i)$
- $i_1, i_2 \leftarrow advancedRead(m_i, z_i)$
- $info \leftarrow concatenate(m_i.flatten(), i_1.i_2, N)$
- $\hat{f} \leftarrow g_{dec}(info)$
- **return** $\hat{f}$

Function `changeShape`$(vector, \mathbb{R}^n, \mathbb{R}^m)$**:**
- change vector's shape from $\mathbb{R}^n$ to $\mathbb{R}^m$
- **return** vector

Function `basicRead`(M, a_i)**:**
- $a_i \leftarrow changeShape(a_i, \mathbb{R}^{d_1 \times d_2}, \mathbb{R}^{d_1 \times d_2 \times 1})$
- $m_i \leftarrow M a_i$
- $m_i \leftarrow changeShape(m_i, \mathbb{R}^{d_1 \times l_z \times 1}, \mathbb{R}^{d_1 \times l_z})$
- **return** m_i

Function `advancedRead`(m_i, z_i)**:**
- $z_i \leftarrow changeShape(z_i, \mathbb{R}^{l_z}, \mathbb{R}^{d_1 \times l_z})$
- $i_1 \leftarrow m_i.min(dim = -1)$
- $z_i^1 \leftarrow where(z_i > \epsilon, z_i, \epsilon)$
- $z_i^2 \leftarrow where(z_i < \epsilon, MAX, 0)$
- $i_2 = [(m_i + z_i^2)/z_i^1].min(dim = -1)$
- **return** i_1, i_2

A.2 Details of meta-task generation

The detailed algorithms for generating basic/adaptive meta-tasks are shown in Algorithm 4 and Algorithm 5, respectively.

Algorithm 4: Generating a Basic Meta-task

Data: Item pool I; Distribution pool P; Frequency mean range L;
Result: a meta-task t_i;

Sample an item size n_i from $[1, |I|]$;
Sample a frequency mean $\bar{f}$ from L;
Sample an subset :$\{x_1^{(i)}, ..., x_{n_i}^{(i)}\}$ of I with size n_i;
Sample a instance $p^{(i)} \sim P$;
for $x_j^{(i)} \in \{x_1^{(i)}, ..., x_{n_i}^{(i)}\}$ **do**
  Sample $p_j^{(i)} \sim p^{(i)}$ and $f_j^{(i)} \leftarrow \lceil n_i \times \bar{f} \times p_j^{(i)} \rceil$;
  add $x_j^{(i)}$ to the t_i's store set (s_i) with $f_j^{(i)}$ times;
  add $(x_j^{(i)}, f_j^{(i)})$ to t_i's query set (q_i);
end

Algorithm 5: Generating an Adaptive Meta-task

Data: Item pool I; Real frequency distribution p; Frequency mean range L;
Result: a meta-task t_i;

Sample an item size n_i from $[1, |I|]$;
Sample a frequency mean $\bar{f}$ from L;
Sample an subset :$\{x_1^{(i)}, ..., x_{n_i}^{(i)}\}$ of I with size n_i;
for $x_j^{(i)} \in \{x_1^{(i)}, ..., x_{n_i}^{(i)}\}$ **do**
  Sample $p_j \sim p$ and $f_j^{(i)} \leftarrow \lceil n_i \times \bar{f} \times p_j \rceil$; `// The correspondence between items and frequencies is changed.`
  add $x_j^{(i)}$ to the t_i's store set (s_i) with $f_j^{(i)}$ times;
  add $(x_j^{(i)}, f_j^{(i)})$ to t_i's query set (q_i);
end

A.3 Hyper-Parameters

We did not deliberately tune the parameters of the meta-sketch. We just followed the setting about conventional NN to choose parameters by balancing the sketching ability and training efficiency. Table 4 shows all hyper-parameters that are considered (best parameters are bolded).

Table 4: Hyper-parameters Considered

Learning rate of MS	$\{1e-3, 5e-4, \mathbf{1e-4}, 5e-5\}$
Hidden Size of g_{emb}	$\{64, \mathbf{128}, 256\}$
Hidden Size of g_{add}	$\{24, \mathbf{48}, 64, 64\}$
Hidden Size of g_{dec}	$\{128, \mathbf{256}, 512\}$
$d_2 : l_r$	$\{5:1, \mathbf{5:2}, 5:4, 5:6\}$
d_1	$\{1, \mathbf{2}, 3\}$

A.4 Ablation Study

As shown in Figure 11, we conduct ablation studies to evaluate some key techniques of the meta-sketch. In all comparisons, the settings follow Section 4 ($n = 5$K,$B = 9$KB ,Word-query), as shown in Table 5. The comparison between Base and Abl 1 shows the effectiveness of the optimizations on operation $\ominus$. The comparison between Base and Abl 2 shows improvement with the address network, especially for the later stages of meta-sketch training. It should be emphasized that embedding vector will pass a $Relu$ activation before the output of the g_{emb}, which allows the model to control the sparsity of embedding vectors easily. In the comparison between Base and Abl 3, we can see the effectiveness of the $Relu$.

Table 5: Settings for the Ablation Study

	Base	Abl1	Abl2	Abl3
$\ominus$	yes	no	yes	yes
g_{add}	yes	yes	no	yes
$Relu$	yes	yes	yes	no

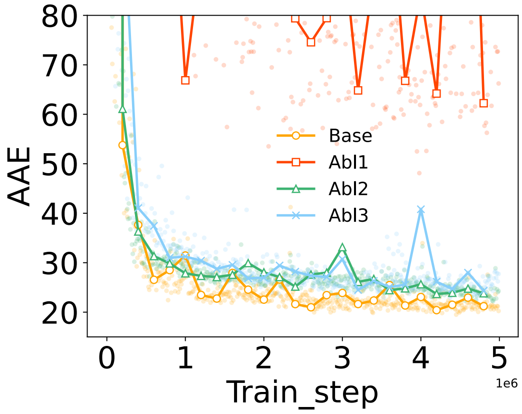

Figure 11: Ablation Study

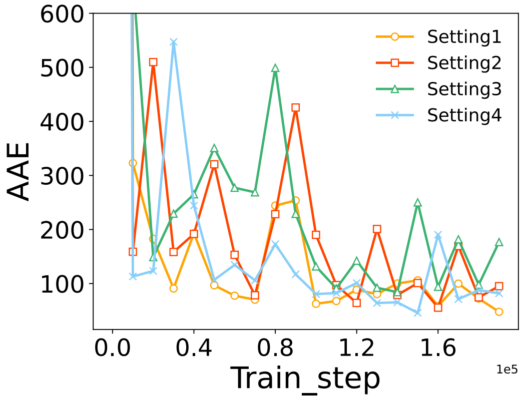

Figure 12: AAE w.r.t. Different Settings of M

A.5 The Default Settings and Discussion for $M(A)$

The default parameters of $M(A)$ under different budgets are shown in Table 6. Note that we first set the size of M, and then obtain the size of the compressed A according to the ratio of $l_z : l_r \approx 5 : 1$.

We further discuss the effect of the setting for M. Figure 12 shows the effect of different settings in Table 7 on the training of the meta-sketch under a fixed 9KB budget. All competitors follow the same training setting ($n = 5$K,$B = 9$KB ,Word-query). For d_1, we set it in the range of 1 to 2, similar to the setting of the number of hash functions in traditional sketches. Figure 12 shows that when $d_1 = 2$, the model yields a better result. For the settings of d_2 and l_z, it shows that the model yields a better result when the ratio of d_2/l_z is around 2. Thus, we set the default parameters for our experiments under the premise of $d_1 = 2$, and $d_2/l_z \approx 2$.

In addition, we can see an inappropriate setting may harm the stability in the early training phase, leading to non-convergence. For example, we can observe that with a additional dimension M corresponds to the better training stability, in the comparison between settings 1 and 4. The comparison of setting 1, setting 2, and setting 3 also shows a reasonable large l_z is beneficial to the stability of the meta-sketch.

Table 6: Default Size of $M(A)$ for Different Space Budgets

B	5KB	7KB	9KB	11KB	13KB	15KB	17KB
d_1	2	2	2	2	2	2	2
d_2	40	45	50	61	61	64	70
l_z	16	20	23	23	27	30	31
l_r	4	4	5	5	5	6	6

Table 7: Settings of M

Setting	1	2	3	4
d_1	2	2	2	1
l_z	23	12	6	34
d_2	50	100	200	68

A.6 The Parameters of Skewed Distributions

Table 8 shows the parameter settings of the three distributions in Section 5 with different skewness levels. Here, the level of skewness is a relative concept under each type of distribution. We can convert all distributions to a zipf form, i.e. sorting n items on a descending order of $\frac{f}{N}$. Afterwards, the level of skewness can be measured by the slope from the first to last positions of the ordering.

Table 8: The Parameters of Skewed Distributions

	Level1	Level2	Level3	Level4
Zipf	$\alpha = 1.0$	$\alpha = 0.8$	$\alpha = 0.6$	$\alpha = 0.4$
Triangular	k=-1/128	k=-1/64	k=-1/32	k=-1/16
Uniform	a=0,b=10000	a=1250,b=8750	a=2500,b=7500	a=3750,b=6250

A.7 Latency and Throughput of Meta-sketch

We evaluate the write/query latency and throughput of the meta-sketch and the CM-sketch[4] under the same setting. We use a single write/query operation for the testing of the latency and a batch of 10K write/query operations for the testing of the throughput. As shown in Table 9, the latency of write/query operations of the meta-sketch is slightly higher than that of the CM-sketch. But with the parallel algebraic operations of NNs, meta-sketch can have a significantly higher throughput, e.g., in GPU environment. For example, the writing throughput of meta-sketch is around $30/1400$ times higher than that of the CM-sketch when deployed on CPU/GPU. Similar observations are drawn on query operations.

Table 9: Latency and Throughput of Meta-sketch

	Write Latency		Query Latency		Write Throughput(QPS)		Query ThroughPut(QPS)	
Device	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
Meta-sketch	0.80ms	1.32ms	1.57ms	2.25ms	166.69k	7142.86k	135.14k	4063.39k
CM-sketch	0.27ms	-	0.25ms	-	4.80k	-	4.86k	-

A.8 Supplementary Experiments

Figure 13 and 14 show the AAEs of different competitors in the experiments of Section 4.2 under different space budgets (B) and different item sizes (n), respectively.

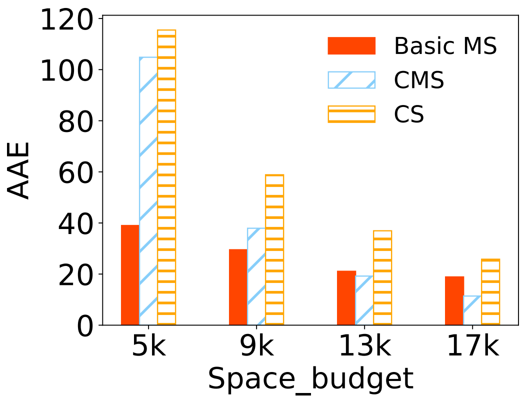

Figure 13: AAE w.r.t. B

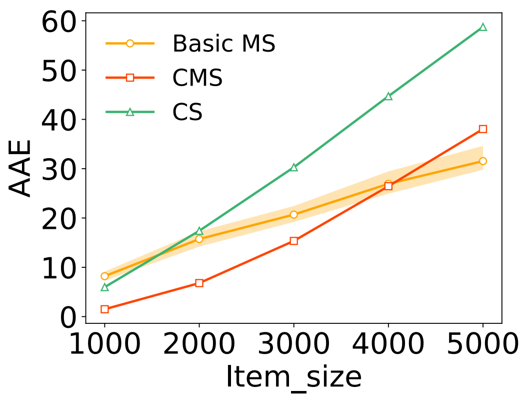

Figure 14: AAE w.r.t. n

A.9 Codes for Meta-sketch

We upload the source code to github: https://github.com/FFY0/meta-sketch.

[4]The implementation of the CM-sketch is from package pyprobables, which is a definitive python library for probabilistic data structures (https://pyprobables.readthedocs.io/en/latest/code.html).