A. Theorems and proofs

A.1. Storage

Bounded storage is an easy desiderata to satisfy.

Claim A.1 *For any $T > 0$, a contextual memory tree after T insertions requires only $O(T)$ storage.*

Proof: The hashmap is $O(T)$. The number of internal nodes is bounded by the number of leaf nodes. Since every leaf node has at least one unique memory, the storage requirement for internal nodes is $O(T)$, and so is the storage requirement for the leaves. □

A.2. Incrementality

By observation, all contextual memory tree algorithms are incremental so the overall operation is incremental as long as the underlying learning algorithms for the learning scorer and routers are incremental. In fact, the contextual memory tree is online so long as the underlying learning algorithms are online.

A.3. Partitioning

Here we prove the partition bound (Theorem 3.3).

Proof: Let R_t and L_t be the number of memories in the right and left subtree respectively, at the start of round t for which we are proving the theorem.

Observe that if

$$\alpha \log \frac{L_t}{R_t} > 1 - \alpha,$$

or, equivalently, if

$$\frac{L_t}{R_t} > e^{\frac{1-\alpha}{\alpha}},$$

or equivalently, if

$$\frac{L_t}{N_t} > \frac{1}{1 + \exp(1 - \frac{1}{\alpha})}, \tag{1}$$

where $N_t = R_t + L_t$ we always have $y = 1$.

A symmetric argument shows that if

$$\frac{R_t}{N_t} > \frac{1}{1 + \exp(1 - \frac{1}{\alpha})}, \tag{2}$$

we always have $y = -1$.

Denote $\kappa = \frac{1}{1+\exp(1-\frac{1}{\alpha})}$. Note that $\kappa < 1$. We claim that for any t, we have

$$R_t \leq (1 - p_t)\kappa N_t + (1 - \kappa) + p_t N_t, \text{ and } L_t \leq (1 - p_t)\kappa N_t + (1 - \kappa) + p_t N_t, \tag{3}$$

where p_t is the progressive training error at the beginning of round t. We prove the claim by induction on t. The base case holds by inspection, assuming $L_2 = R_2 = 1$ and $p_2 = 0$ (i.e., by simply initializing all leaf with a default example).

Assume that the claim holds for step t, and consider step $t + 1$.

Below we first consider the first case: (1) $R_t > \kappa N_t$.

Note that in this case, we always have $y = -1$. Whether or not we route the example to the left depends on whether or not the router makes a post-update mistake. Hence, we discuss two sub-cases below.

(a) The router does not make a mistake here. In this case, the router routes the example to the left. Since no mistake happens in this round, we have $p_{t+1}N_{t+1} = p_t N_t$, i.e., the total number of mistakes remain the same. Then, we have:

$$R_{t+1} = R_t \leq (1 - p_t)\kappa N_t + (1 - \kappa) + p_t N_t \leq (1 - p_{t+1})\kappa N_{t+1} + (1 - \kappa) + p_{t+1} N_{t+1}, \tag{4}$$

where the inequality comes from the fact that $N_{t+1} = N_t + 1 > N_t$.

Now we consider the second sub-case here.

(b) The router does make a mistake. In this case, the router routes the example to the right. Note that in this case, we have $p_{t+1}N_{t+1} = p_tN_t + 1$, i.e., the total number of mistakes increases by one. Hence, we have for R_{t+1}:

$$\begin{aligned} R_{t+1} &= R_t + 1 \le (1-p_t)\kappa N_t + (1-\kappa) + p_tN_t + 1 \\ &= \kappa N_t - \kappa p_t N_t + (1-\kappa) + p_tN_t + 1 \\ &= \kappa N_t - \kappa p_{t+1}N_{t+1} + \kappa + (1-\kappa) + p_tN_t + 1 \\ &= \kappa N_{t+1} - \kappa p_{t+1}N_{t+1} + (1-\kappa) + p_{t+1}N_{t+1}, \end{aligned} \tag{5}$$

where the second equality uses the fact that $\kappa p_{t+1}N_{t+1} = \kappa p_tN_t + \kappa$.

With case (a) and case (b), we can conclude that for case (1) where $R_t > \kappa N_t$, we have:

$$R_{t+1} \le (1-p_{t+1})\kappa N_{t+1} + (1-\kappa) + p_{t+1}N_{t+1}. \tag{6}$$

Now we consider the second case (b): $R_t \le \kappa N_t$. In this case, regardless of where the example routes, we always have:

$$R_{t+1} \le R_t + 1 \le \kappa N_t + 1 = \kappa(N_{t+1} - 1) + 1 = \kappa N_{t+1} + 1 - \kappa. \tag{7}$$

Note that since $\kappa < 1$, we must have $p_{t+1}N_{t+1} \ge p_{t+1}\kappa N_{t+1}$. Hence we have

$$\begin{aligned} R_{t+1} &\le \kappa N_{t+1} + 1 - \kappa \\ &\le \kappa N_{t+1} + 1 - \kappa + p_{t+1}N_{t+1} - \kappa p_{t+1}N_{t+1} = (1-p_{t+1})\kappa N_{t+1} + (1-\kappa) + p_{t+1}N_{t+1}. \end{aligned} \tag{8}$$

With case (1) and case (2), we can conclude that for R_{t+1}, we always have:

$$R_{t+1} \le (1-p_{t+1})\kappa N_{t+1} + (1-\kappa) + p_{t+1}N_{t+1}. \tag{9}$$

A symmetric argument implies

$$L_{t+1} \le (1-p_{t+1})\kappa N_{t+1} + (1-\kappa) + p_{t+1}N_{t+1}. \tag{10}$$

By induction, we prove our claim.

Now given $L_t \le (1-p_t)\kappa N_t + (1-\kappa) + p_tN_t$, we divide N_t on both sides to get:

$$L_t/N_t \le (1-p_t)\kappa + \frac{1-\kappa}{N_t} + p_t. \tag{11}$$

Multiplying both sides by -1 and adding 1, we get:

$$\begin{aligned} 1 - L_t/N_t = \frac{R_t}{N_t} &\ge 1 - (1-p_t)\kappa + \frac{\kappa - 1}{N_t} - p_t \\ &= (1-p_t) - (1-p_t)\kappa + \frac{\kappa-1}{N_t} \\ &= (1-p_t)(1-\kappa) + \frac{\kappa-1}{N_t}. \end{aligned} \tag{12}$$

As $\kappa > 0$, we get:

$$R_t/N_t \ge (1-p_t)(1-\kappa) - \frac{1}{N_t}. \tag{13}$$

By symmetry, we have:

$$L_t/N_t \ge (1-p_t)(1-\kappa) - \frac{1}{N_t}. \tag{14}$$

Substituting κ in, we get:

$$\min\{L_t/N_t, R_t/N_t\} \ge (1-p_t)\frac{1}{\exp(\frac{1-\alpha}{\alpha}) + 1} - \frac{1}{N_t}. \tag{15}$$

□

A.4. Depth of K-partitions

Next we prove a depth bound given K-partitions.

Lemma A.2 *A tree on T points with a K-partition at every internal node has depth at most $K \log T$.*

Proof: By assumption, each internal node routes at least a $1/K$ fraction of incident points in either direction, hence at most a $1 - 1/K$ fraction of points are routed the other direction. As a consequence, at a depth d a node has at most $t(1 - 1/K)^d$ memories beneath it. The deepest internal node in the tree satisfies:

$$T(1 - 1/K)^d \geq 1$$

rearranging, we get:

$$T \geq \left(\frac{1}{1 - 1/K}\right)^d$$

Taking the log of both sides, we get:

$$\log T \geq d \log \left(\frac{1}{1 - 1/K}\right)$$

which implies

$$d \leq \frac{\log T}{\log \left(\frac{1}{1-1/K}\right)}.$$

Using $-\log(1 - x) \geq x$ for $0 \leq x < 1$, we get

$$d \leq K \log T.$$

□

A.5. Computational bound proof

Now we prove Theorem 3.4.

Proof: We assume that d is constant. From the depth bound, REMOVE is $O(K \log T)$. INSERTLEAF is $O(1)$ if the guard on line 2 is false. If the guard is true, then we know that $|v.m| > c \log T$ and $|v.m| - 1 \leq c \log T$ since otherwise it would have been triggered on a previous insertion. Hence, line 5 executes $O(c \log T)$ times, with each invocation of INSERT taking $O(1)$ time in this case as the while loop in line 1 is executed only once.

INSERT$(\cdot, \cdot, 0)$ takes $O((K + c) \log T)$ from the depth bound and the complexity of INSERTLEAF. Thus the computational complexity of REROUTE is $O((K + c) \log T)$. UPDATE takes $O(1)$ time, followed by d invocations of REROUTE, making it $O((K + c) \log T)$ time as well. INSERT$(\cdot, \cdot, d)$ takes $O((K + c) \log T)$, followed by d invocations of REROUTE, making its total complexity $O((K + c) \log T)$.

QUERY calls PATH at most twice and then pays $O(c \log T)$ computation to find the top k memories for the query. The complexity of PATH is $O(K \log T)$, making the overall complexity of QUERY $O((K + c) \log T)$. □

A.6. Self-Consistency

Let us recall the definition of self-consistency.

Definition A.3 *A* CMT *is* **self-consistent** *if for all z with a unique $z.x$, $z =$*QUERY$(z.x, 1, 0)$.

It is easy to see that self-consistency holds for any z immediately after insertion.

Lemma A.4 *If $z = \arg\max_{z'} f(z.x, z')$, then $z =$*QUERY$(z.x, 1, 0)$ *immediately after* INSERT$(\text{root}, z, 0)$.

Proof: By construction, the updates in line 4 of INSERT do not affect the routers at nodes closer to the root. Therefore, since INSERT line 6 and PATH line 3 are identical, both INSERT and QUERY walk through the same internal nodes. At INSERTLEAF, the last execution of line 5 is for z and hence any newly created internal node also routes in the same direction. Once a leaf is reached, $z = \arg\max_{z'} f(z.x, z')$ implies the claim follows. □

Achieving self-consistency for all z simultaneously is more difficult since online updates to routers can invalidate pre-existing self-consistency. Nevertheless, the combination of the REROUTE operation and the convergence of learning algorithms at internal nodes leads to asymptotic self-consistency.

Definition A.5 *A* **convergent learning algorithm** *satisfies, for all input distributions D and all update sequences,*

$$\mathbf{P}_{x \sim D}[g_t(x) \neq g_{t-1}(x)] = 0,$$

in the limit as $t \to \infty$.

Restated, a convergent learning algorithm is one that disturbs fewer predictions the more updates that it gets. This property is an abstraction of many existing update rules with decaying learning rates.

Theorem A.6 *For all contextual memory trees T, if $z = \arg\max_{z'} T.f(z.x, z')$ for all z, and all routers g are convergent under the induced sequence of updates, then in the limit as $T.d \to \infty$, T is self-consistent almost surely.*

Proof: The proof operates level-wise. The uniform REROUTE operation and the fact that the learning algorithm at the root is convergent by assumption guarantees that the root eventually routes in a self-consistent fashion almost surely. Once the root converges, the same logic applies recursively to every internal node, for the distribution of memories induced at the node. To finish the proof, we just use the assumption that $z = \arg\max_{z'} f(z.x, z')$. □

Asymptotic self-consistency is a relatively weak property so we also study self-consistency empirically in section 4.5.

A.7. Learning

Finding a good partition from a learning perspective is plausibly more difficult than finding a good classifier. For example, in a vector space finding a partition with a large margin which separates input points into two sets each within a constant factor of the original in size is an obvious proxy. The best results for this problem (Xu & Schuurmans, 2005; Karnin et al., 2012) do not scale to large datasets or function in an online fashion.

For any given node we have a set of incident samples which cause updates on INSERT or UPDATE. Focusing on UPDATE at a single node, the natural function to optimize is a form of balanced expected reward. If r_a and p_a are the rewards and probabilities of taking action a, then a natural objective is:

$$\arg\max_g E_{x\sim D}(1-\alpha) r_{g(x)} - \alpha \log p^g_{g(x)} \tag{16}$$

where $p^g_a = \Pr_{x\sim D}(g(x) = a)$ is the probability that g chooses direction a as induced by samples over x. This objective both maximizes reward and minimizes the frequency of the chosen action, implying a good solution sends samples in both directions.

The performance of the partitioner is dependent on the classifier g which optimizes importance weighted binary classification. In particular, we evaluate the performance of g according to:

$$\hat{E}_{x,y,i} i I(g(x) \neq y)$$

with the goal of g minimizing the empirical importance weighted loss over observed samples.

Next we prove a basic sanity check theorem about the asymptotics of learning a single node. For this theorem, we rely upon the notion of a no-regret (Cesa-Bianchi & Lugosi, 2006) g which is also convergent. Common no-regret algorithms like Hedge (Freund & Schapire, 1997) are also convergent for absolutely continuous D generating events. The following theorem relies on the

Theorem A.7 *For all absolutely continuous distributions D over updates with $d = 0$ reroutes and for all compact convergent no-regret g:*

$$\lim_{t\to\infty} g_t$$

exists and is a local maxima of (16).

The proof is in Appendix A.8. Here, convergent g is as defined in section A.6 and compact g refers to the standard definition of a compact space for the parameterization of g.

It's important to note that the $d = 0$ requirement is inconsistent with the $d \to \infty$ requirement for self-consistency. This tradeoff is fundamental: a learning process that is grounded in unsupervised updates (as for self-consistency) is fundamentally different from a learning process grounded in rewards (as for the learning update). If these two groundings happen to agree then compatibility exists as every unsupervised update is consistent with a reward update.

This theorem shows that the optimization process eventually drives to a local maxima of (16) providing a single node semantics. Since every node optimizes independently, the joint system therefore eventually achieves convergence over 1-step routing deviations.

A.8. Learning proof

Proof: Consider without loss of generality the root node of the tree, and then apply this argument recursively.

Since g is no-regret the g minimizing (16) for any observed p eventually wins. Since the D producing updates is absolutely continuous, convergence of g implies convergence of p and the g, p system is compact since g is compact and p is compact. Given this, a g, p pair maps to a new g, p pair according to the dynamics of the learning algorithm.

Brouwer's fixed point theorem (Brouwer, 1911) hence implies that there exists a g, p pair which is a fixed point of this process. Since g is no-regret, the system must eventually reach such a fixed point (there may be many such fixed points in general).

For a given g, let $\Phi^{(g)} = E_{x\sim D}\left[(1-\alpha)r_{g(x)} - \alpha \log p^g_{g(x)}\right]$ be the objective in equation (16) and define

$$r^{(g)}_{\text{Left}} = (1-\alpha)r_{\text{Left}} - \alpha \log p^g_{\text{Left}}$$
$$r^{(g)}_{\text{Right}} = (1-\alpha)r_{\text{Right}} - \alpha \log p^g_{\text{Right}}.$$

Using this definition, we can define:

$$\begin{aligned} y^{(g)} &= r^{(g)}_{\text{Right}} - r^{(g)}_{\text{Left}} \\ &= (1-\alpha)(r_{\text{Right}} - r_{\text{Left}}) - \alpha(\log p^g_{\text{Right}} + \log p^g_{\text{Left}}) \\ &= (1-\alpha)(r_{\text{Right}} - r_{\text{Left}}) + \alpha \log \frac{p^g_{\text{Left}}}{p^g_{\text{Right}}} \\ &\overset{a.s.}{=} (1-\alpha)(r_{\text{Right}} - r_{\text{Left}}) + \alpha \lim_{\substack{t\to\infty \\ S\sim D^t}} \log \frac{\sum_{x\in S} I(g(x) = \text{Left})}{\sum_{x\in S} I(g(x) = \text{Right})}. \end{aligned}$$

Assume wlog that $r^{(g)}_{\text{Right}} > r^{(g)}_{\text{Left}}$ such that $|y^{(g)}| = y^{(g)}$. Examining Line 5 of UPDATE, for a fixed g (i.e. g.update() has converged), taking expectations wrt p over a, and denoting H as the complete empirical history of the node,

$$\begin{aligned} E_p y &= (1-\alpha)E_{a\sim\vec{p}}\left(\frac{r_{\text{Right}} I(a = \text{Right})}{p(\text{Right})} - \frac{r_{\text{Left}} I(a = \text{Left})}{p(\text{Left})}\right) + E_{x\sim D} \log \frac{\sum_{x\in H} I(g(x) = \text{Left})}{\sum_{x\in H} I(g(x) = \text{Right})} \\ &\overset{t\to\infty}{=} (1-\alpha)(r_{\text{Right}} - r_{\text{Left}}) + E_{\substack{x\sim D \\ S\sim D^t}} \log \frac{\sum_{x\in S} I(g(x) = \text{Left})}{\sum_{x\in S} I(g(x) = \text{Right})}. \end{aligned}$$

In other words, $\lim_{t\to\infty} E_p y \overset{a.s.}{=} y^{(g)}$. The expected loss of g then converges to:

$$E\left[|y^{(g)}| I(g(x) \neq \text{sign}(y^{(g)}))\right] \overset{a.s.}{=} \Phi^{(g)}$$

proving the theorem. □

B. Experimental Details

B.1. Datasets

dataset	task	classes	examples
ALOI	Visual Object Recognition	10^3	10^5
WikiPara (S-shot)	Language Modeling	10^4	$S\times10^4$
ImageNet (S-shot)	Visual Object Recognition	2×10^4	$2S\times10^4$
Pascal	Image-Caption Q&A	/	10^3
Flickr-8k	Image-Caption Q&A	/	8×10^3
MS COCO	Image-Caption Q&A	/	8×10^4

Table 3: Datasets used for experimentation on multi-class and Image Retrieval

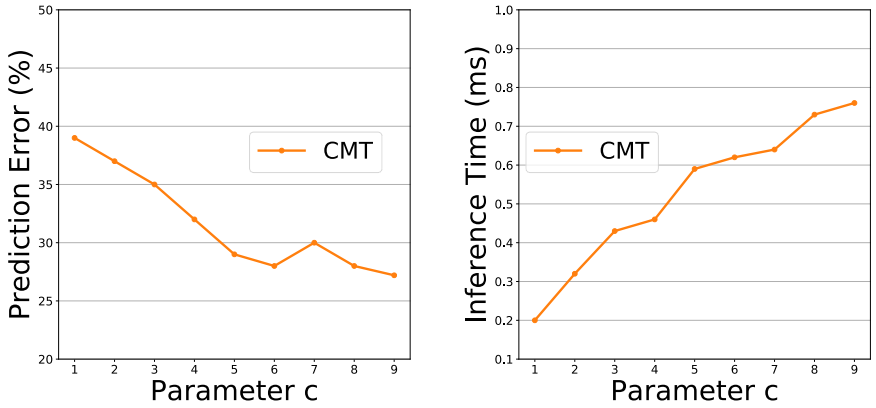
dataset	# Training	# test	# Categories	# Features	Avg # Points/Label	Avg # Labels/Point
RCV1-2K	623847	155962	2456	47236	1218.56	4.79
AmazonCat-13K	1186239	306782	13330	203882	448.57	5.04
Wiki10-31K	14146	6616	30938	101938	8.52	18.64

Table 4: Extreme Multi-Label Classification datasets used for experimentation

Table 3 summarizes the datasets used in Multi-class classification and image retrieval experimentations. ALOI (Geusebroek et al., 2005) is a color image collection of one-thousand small objects. We use the same train and test split and feature representation as Recall Tree

	# unsupervised passes	# supervised passes	c	d	α
ALOI	1	2	4	5	0.1
Few-shot WikiPara	1	1	4	5	0.9
Few-shot ImageNet	1	1	4	3	0.9
RCV1-1K	1	3	2	3	0.9
AmazonCat-13K	1	3	2	3	0.9
Wiki10-31K	1	3	2	3	0.9
Pascal	1	1	10	1	0.9
Flickr	1	1	10	1	0.9
MS COCO	1	1	10	1	0.9

Table 5: Key parameters used for CMT for our experiments



(a) Prediction Error (b) Inference Time

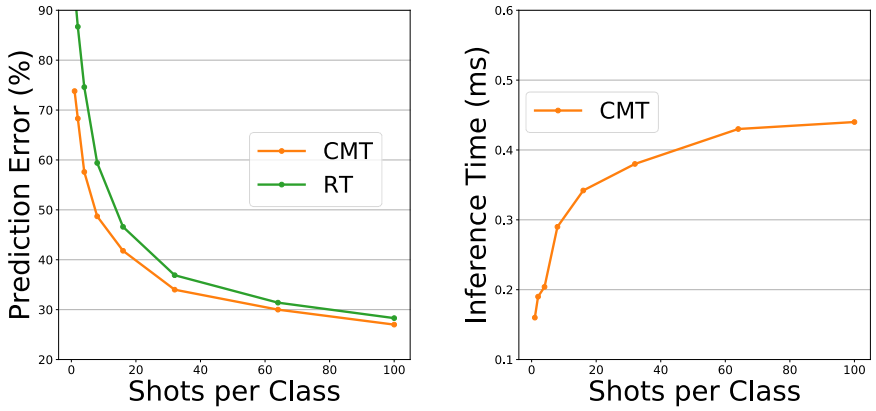
Figure 6: Performance and inference time of CMT versus the number of examples per leaf

(Daumé et al., 2017). The few-shot ImageNet datasets are constructed from the whole ImageNet that has 20,000 classes and 10^7 training examples. We use the same train and test split as Recall Tree (Daumé et al., 2017). The features of images are extracted from intermediate layers of a convolutional neural network trained on the ILVSRC2012 (Oquab et al., 2014). To construct a S-shot ImageNet dataset, we randomly sample S training examples for each class. A S-shot ImageNet dataset hence has a $20000 \times S$ many training examples.

Pascal sentence dataset consists of 1000 pairs of image $I \in \mathbb{R}^{300\times180}$ and the corresponding description of the image. We compute HoG feature y for each image I and token occurrences $y \in \mathbb{R}^{2^{20}}$ for each description using Scikit-learn's Hashing functionality. The resulting feature x is high dimensional but extremely sparse. We randomly split the dataset into a training set consisting of 900 pairs of images and their descriptions and a test set with the remaining data. A memory $z = (x, y)$ here consists of the image feature y and the descriptions' feature x. During inference time, given a query x (i.e., a description of an unknown image), CMT retrieves a memory $z' = (x', y')$, such that the image y' associated with the memory z' is as similar to the unknown image associated with the test query x. Given two memories z and z', the reward signal is defined as $r(z_y, z'_y) = \langle z_y, z'_y \rangle$. The Flickr8k dataset consists of 8k images and 5 sentences descriptions for each image. Similar to Pascal, we compute HoG feature y for each image and hashing feature x for its 5-sentence description. The MS COCO image caption dataset consists of 80K images in training set, 4000 images in validation set and testing set. We extract image feature y from a fully connected layer in a VGG-19 (Simonyan & Zisserman, 2014) pre-trained on ILSVRC2012 dataset . We use hashing feature x for image captions.

Table 4 summarizes the datasets used for multi-label classification task. All three datasets are obtained from the Extreme Classification Repository (http://manikvarma.org/downloads/XC/XMLRepository.html).

All datasets that we used throughout this work are available at (url will be provided here).



(a) Prediction Error versus Shots

(b) Inference Time verus Shots

Figure 7: Performance (a) and inference time (b) of CMT versus the number of training examples per label in ALOI (i.e., shots S).

B.2. Extra Plots in Sec. 4.5

Figure 6 shows the detailed plots of CMT's statistical performance (a) and inference performance (b) with respect to parameter c (i.e., the maximum number of memories per leaf: $c \log(N)$). As shown in Figure 6 (b), the inference time increases almost linear with respect to c, which is expected as once we reach a leaf, we need to scan all memories stored in that leaf.

Figure 7 shows the detailed plots of CMT's statistical performance (a) and inference time (b) with respect to the number of shots (i.e., number of training examples for each class) in ALOI. Note that ALOI has in total 1000 classes and hence for ALOI S-shot, we will have in total $S \times 1000$ examples. Namely as S increases, CMT has more memories to store. We vary S from 1 to 100. Figure 7a shows the performance of CMT improves quickly as S increases (e.g., dataset becomes easier to learn). Also CMT consistently outperform Recall Tree, with larger margin at fewer shots. From Figure 7 (b), we see that the inference time increases sublinearly with respect to the number of shots (i.e., the number of total memories stored in CMT), which is also expected, as we show that the depth of CMT and the number of memories per leaf are logarithmic with respect to the size of CMT.

B.3. Few-shot Extreme Multi-class Classification Details

Table 6 shows the detailed prediction error and inference time of CMT and other baselines. For ALOI, we briefly tuned the parameters of CMT based on a set of holdout training data, and for few-shot WikiPara (and few-shot ImageNet), we briefly tuned the parameters of CMT using the one-shot dataset on hold-out dataset and then simply just use the same set of parameters across all other few-shot datasets. The detailed key parameters can be found in Table 5. Note that the parameters c (leaf memories multiplier), d (number of REROUTE calls per insertion), and α (regularization parameter to ensure balance of CMT) are the tthree key extra parameters we have compared to the baselines considered here such as Recall Tree and LOMTree.

One interesting observation from Table 6 is that CMT can outperform even OAA at the one-shot WikiPara experiment. All the datasets have same number of examples per class and hence a constant predictor (i.e., prediction by majority) would have prediction accuracy at $1/$(# of classes). In terms of computation, due to the overhead of storing memory and dynamically allocating memory in CMT, CMT in general is less computationally efficient than other logarithmic baselines (LOMTree & Recall Tree). Comparing to highly optimized implementation of OAA in VW, we observe that CMT is less computationally efficiently on smaller dataset such aas ALOI, while for datasets with extremely large number of labels, CMT consistently outperform OAA in terms of computation efficiency.

B.4. Multi-Label Classification

The key parameters we used to conduct our multi-label experiments are summarized in Table 5. We briefly tuned the number of supervised passes and α on holdout training datasets and picked a set of parameters that worked well for all datasets in general. We did not tuned parameters c and d. The results are summarized in table 1.

B.5. Image Retrieval

For Pascal and Flickr8k, we randomly split the dataset into a pair of training set and test set. We create 5 random splits, and use one split for tuning parameters for CMT. For MS COCO, we use the default training, validation, and test split, and tune parameters on validation set.

		CMT (u)	CMT	LOMTree	Recall Tree	OAA
ALOI	Test Error	75.8	26.3	66.7	28.8	21.7
	Test Time	0.27	0.15	0.01	0.02	0.05
WikiPara (1-shot)	Test Error	97.3	96.7	98.2	97.1	98.2
	Test Time	0.3	0.3	0.1	0.1	0.9
WikiPara (2-shot)	Test Error	96.3	96.0	96.7	94.0	95.6
	Test Time	0.4	0.4	0.1	0.1	1.1
WikiPara (3-shot)	Test Error	96.1	95.7	96.1	92.0	92.8
	Test Time	0.5	0.3	0.1	0.1	1.1
ImageNet (1-shot)	Test Error	98.8	98.7	99.8	99.7	98.0
	Test Time	9.6	8.2	1.0	3.3	112.4
ImageNet (2-shot)	Test Error	98.7	98.3	99.6	99.3	97.0
	Test Time	11.7	8.6	1.2	3.3	112.0
ImageNet (3-shot)	Test Error	98.6	98.1	99.4	98.9	96.2
	Test Time	9.8	8.5	4.6	3.3	109.0
ImageNet (5-shot)	Test Error	98.4	97.9	99.2	98.6	95.3
	Test Time	12.5	11.6	1.3	4.0	110.4

Table 6: Prediction error (%) and inference time (ms) of different mult-class classification algorithms on few-shot extreme multi-class classification datasets.

		CMT (u)	CMT	NN	KD-Tree w/ PCA
Pascal	Test Reward	0.680±0.008	0.694±0.010	0.683 ±0.013	0.675 ±0.013
	Test Time (ms)	0.13	0.58	0.74	0.002
Flickr8k	Test Reward	0.733±0.004	0.740±0.002	0.736 ±0.003	0.733 ± 0.002
	Test Time (ms)	0.23	1.0	6.0	0.002
MS COCO	Test Reward	0.581	0.584	0.585	0.574
	Test Time (ms)	0.590	1.90	12.4	35.4

Table 7: Performance (average reward % and time ms) of different approaches on image retrieval tasks.

For image retrieval applications, the key parameters used by CMT are summarized in Table 5, and the detailed performances of CMT and NN are summarized in Table 7. For Pascal and Flickr8k, since we have 5 training/test split, we report mean and standard deviation.

In this set of experiments, for CMT, during training we set k to be $c\log(N)$, i.e., we returned all memories stored in a single leaf to get reward signals to update f. During testing, for both CMT and NN, we report the average reward of the top returned memory on given test sets.

Table 7 summarizes the performance of CMT, NN and KD-Tree operating on a low dimension feature of the query computed from the randomized PCA algorithm from sklearn. We choose the reduced dimension of the feature such that the total PCA time plus the KD-Tree construction time is similar to the time of unsupervised CMT construction time (in Pascal, the reduced dimension is 20; in Flickr8k, the reduced dimension is 200; in MSCOCO, the reduced dimension is 200). Note that on Pascal and Flickr8k, CMT slightly outperforms NN in terms of average reward on test sets, indicating the potential benefit of learned memories. CMT statistically outperforms KD-tree operated on the low dimensional feature computed from PCA.