Token Turing Machines are Efficient Vision Models

Purvish Jajal
Purdue University

Nick John Eliopoulos
Purdue University

Benjamin Shiue-Hal Chou
Purdue University

George K. Thiravathukal
Loyola University Chicago

James C. Davis
Purdue University

Yung-Hsiang Lu
Purdue University

Abstract

We propose Vision Token Turing Machines (ViTTM), an efficient, low-latency, memory-augmented Vision Transformer (ViT). Our approach builds on Neural Turing Machines (NTM) and Token Turing Machines (TTM), which were applied to NLP and sequential visual understanding tasks. ViTTMs are designed for non-sequential computer vision tasks such as image classification and segmentation. Our model creates two sets of tokens: process tokens and memory tokens; process tokens pass through encoder blocks and read-write from memory tokens at each encoder block in the network, allowing them to store and retrieve information from memory. By ensuring that there are fewer process tokens, we are able to reduce the inference time of the network while maintaining its accuracy. On ImageNet-1K, the state-of-the-art ViT-B has median latency of 529.5ms and 81.0% accuracy, while our ViTTM-B is 56% faster (234.1ms), with 2.4× fewer FLOPs, with an accuracy of 82.9%. On ADE20K semantic segmentation, ViT-B achieves 45.65mIoU at 13.8 frame-per-second (FPS) whereas our ViTTM-B model acheives a 45.17 mIoU with 26.8 FPS (+94%).

1. Introduction

Vision Transformers (ViTs) [8] are used for a variety of vision tasks such as image recognition, semantic segmentation, object detection, and even image generation. ViTs are a building block for many applications and models, as both backbones and foundation models. However, high-performing ViTs incur large computational costs due to their quadratic complexity with respect to input size. Computational costs can be reduced by processing fewer tokens. Methods such as token sparsification remove uninformative tokens during inference [1, 28]. However, sparsification compromises accuracy and fine-tuning or architectural
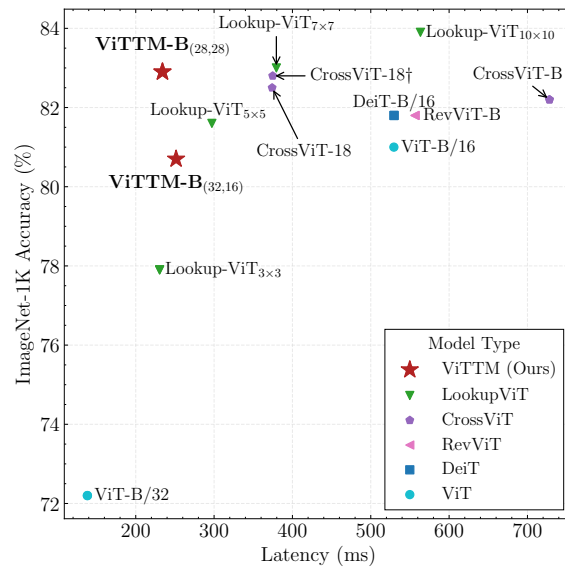

Figure 1. Comparison of our architecture with state-of-the-art methods. It is evident that ViTTM-B$_{(28,28)}$ has lower latency than its accuracy equivalents (*e.g.* Lookup-ViT$_{7\times7}$) while having higher accuracy than its latency equivalents (*e.g.* Lookup-ViT$_{3\times3}$).

modifications are required to recover accuracy.

Architectural components such as memory mechanisms can improve accuracy in various tasks [14, 27, 39]. They are fixed size, and prior work has effectively used memory to enhance models on sequential language and algorithmic tasks [14]. ViTs trained with memory can significantly improve their accuracy [33]. Recently, Token Turing Machines (TTMs) [31] have employed memory with Video Transformers to efficiently process video. LookupViT [21] uses lookup tokens (akin to memory) to efficiently process images.

In this work, we present ViTTM, a NTM-ViT hybrid that integrates memory within ViT on a per-layer basis, creating
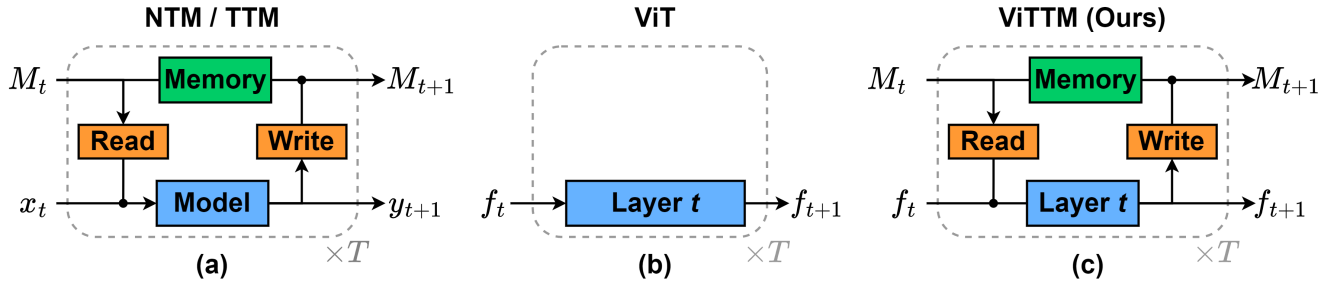
Figure 2. Comparison of NTM/TTMs, ViTs, and ViTTMs. *(a)* NTMs are sequential models that process an input sequence of size T, where inputs x_t are processed at each time step t and the memory M is read from and written at each time step. *(b)* ViTs process a single input $f_0 (= x_0)$, through a series of T layers, where each layer is indexed by t, the output features of each layer are denoted f_t. Our ViTTMs are a synthesis of the NTM and ViT architectures. ViTTMs integrate memory into the ViT architecture on a per-layer basis, processing a sequence of features f_t rather than input sequences.

an architecture that is efficient (low latency) and accurate (Fig. 2). Unlike NTMs, ViTTM processes a sequence of features, not input sequences. By employing a fixed size memory, ViTTM reduces the number of tokens processed through each layer compared to ViTs. Integrating memory ensures that ViTTM is *efficient* by processing fewer tokens and *accurate* by storing pertinent information in memory.

We evaluate ViTTM on the ImageNet-1K dataset and the ADE20K semantic segmentation dataset. ViTTM has competitive accuracy-latency tradeoffs on ImageNet1K classification and expands the Pareto front. Our ViTTM-B, comparable with ViT-B, requires 2.4× fewer FLOPs, reducing latency by 56% while having a 1.9% higher accuracy on ImageNet-1K (82.9%) than ViT-B. Our model is also competitive on ADE20K semantic segmentation. Compared with ViT-B, ViTTM-B achieves an mIoU of 45.17 (−0.48) while achieving 94% higher FPS on an NVIDIA A30 GPU.

We make the following contributions:

- We present the ViTTM architecture, an efficient memory-augmented ViT that has competitive accuracy-latency trade-offs on classification and semantic segmentation (Sec. 3.2).
- We conduct a comprehensive ablation of the ViTTM architecture, exploring various designs and identify key decisions that impact the quality of the architecture (Sec. 4.2).

2. Related Work

In this section we present prior work in ViTs, token-sparsification, and memory-augmented neural networks. Sec. 2.1 presents prior work in ViTs and highlight their differences with ViTTM. We present prior work in token sparsification in Sec. 2.2 and highlight the benefits of ViTTM over sparsification. Prior work in memory-augmented neural networks (MANNs) is presented in Sec. 2.3 and we highlight how we adapt MANNs to non-sequential tasks.

2.1. Vision Transformers

After transformers achieved success in language processing tasks [38], researchers adapted transformer models for computer vision tasks. ViT [8] shows that transformers are effective on computer vision tasks. Subsequent works have focused on developing: (1) training regimes that improve ViT task performance, and (2) alternative ViT architectures to achieve better accuracy-latency trade-offs.

First, alternative training recipes and self-supervised training are effective at improving ViTs. Touvron *et al.* show that ViTs can achieve high accuracy by using data augmentation strategies, without using large datasets [36, 37]. Steiner *et al.* conduct an in-depth empirical study on the training of ViTs [34]. They highlight the interplay between *compute*, *data*, *model size*, and *regularization* for the effective training of ViTs. In addition, self-supervised learning (SSL) has become a feature of ViT training [2, 4, 17].

Second, a variety of ViT architectures have been proposed that offer different, or strictly better, accuracy-latency trade-offs. These include hierarchical ViTs like SwinTransformer and MViT, which create multi-scale feature maps, and CNN-ViT hybrids that incorporate convolutional biases [11, 23]. Among these are *two-stream* architectures — our work is most similar to these. Cross-ViT [3] one of the first two-stream ViT designs, processes tokens at different patch sizes in separate branches and then merges them through an efficient cross-attention mechanism. ViT-CoMer [42] utilizes bi-directional interactions with CNNs using a fusion block. Reversible Vision Transformers [24] create a two-stream architecture with the goal of decoupling GPU memory requirements from model depth by reducing activation caching during training. A contemporaneous work, LookupViT [21], processes inputs using two sets of tokens, higher resolution (lookup) tokens and compressed tokens, with information being exchanged between them.

In this work we present a novel two-stream architecture. Unlike prior two-stream architectures, we conduct the majority of the computations in a lightweight stream, thus being computationally efficient. We highlight differences between our work and prior work in Sec. 3.1. The closest work, LookupViT is conceptually similar but differs in design.

2.2. Token Sparsification

Token sparsification is a technique used to accelerate ViTs by reducing the overall number of tokens, which also reduces compute cost. However, sparsification will degrade accuracy without a good heuristic to distinguish informative and uninformative tokens [25]. Sparsification methods can be broadly categorized into two approaches: *pruning* methods and *merging* methods [16].

Pruning-based methods identify and remove uninformative tokens during inference, thereby reducing computation costs. Popular approaches include using prediction modules [28], or using intermediate attention computations as a heuristic to identify unimportant tokens [12, 20, 22, 43].

Merging-based methods reduce the total token count by identifying and combining similar tokens. Boyla *et al.* propose a training-free merging approach, which averages similar tokens at every layer [1]. Renggli *et al.* uses learned modules to merge tokens at a single layer, reducing tokens of up to $24\times$ and consequently large reductions in computational costs [29]. Similary, Ryo *et al.* improve ViTs on video tasks by learning to merge tokens [32].

Similar to token sparsification, our ViTTM architecture processes fewer tokens at a time by increasing patch size though our approach is not necessarily analogous to pruning or merging methods. Unlike sparsification, we have access to a memory unit that mitigates the accuracy degradation associated with removing tokens.

2.3. Memory Augmented Neural Networks

Memory Augmented Neural Networks (MANNs) are an extension to neural networks that enhance their capabilities with memory [27]. Originally, MANNs enabled RNNs to better tackle various sequential algorithmic, language, and reasoning tasks. Subsequently, memory has been used to enhance neural networks in various vision tasks.

Neural Turing Machines are one of the first MANNs [14]. NTMs consist of an RNN equipped with a set of read-write heads that interact with an external memory bank, these modifications allow NTMs to outperform RNNs in copy, sorting, and algorithmic tasks. Memory Networks are a concurrent approach that focuses upon language and reasoning tasks [39]. Differentiable Neural Computers (DNCs) further extend these NTMs by adding dynamic memory allocation [15]. Similar to prior works, DNCs show effectiveness on a variety of algorith-

mic, language, and reasoning tasks. Recent works, augment transformers with memory sequence modelling [41].

Recent work has adapted MANNs to both sequential and non-sequential vision tasks. ViTs have been augmented with learnable memory tokens at every layer to enhance fine-tuning [33]. Token Turing Machines (TTM) extend NTMs to transformer models for sequential visual understanding tasks, such as video activity detection [31]. TTM uses an external memory for reading and writing, and introduce a token summarization module that maintains constant computational cost regardless of sequence length. Our insight is that NTMs can be extended to ViTs by treating intermediate features as a sequence. In contrast to prior work, we create an architecture that uses memory for the processing of non-sequential tasks.

3. Vision Token Turing Machines (ViTTM)

We begin by providing the intuition behind ViTTMs and how this work is related to prior work and draw parallels with NTMs in Sec. 3.1. Then, we elaborate on the ViTTM architecture in Sec. 3.2.

3.1. Intuition and Comparison to Prior Work

The ViTTM architecture is inspired by two key insights: (1) NTMs and TTMs have shown that neural networks can learn to use memory [15, 31]; (2) *fewer tokens* through transformer blocks makes inference *faster*, and *more tokens* mean *better accuracy* [25]. By integrating these insights, ViTTMs are able to process only a few tokens at a time (*efficiency*), while being able to access the information stored in many tokens (*accuracy*) by using memory. The end result is an architecture that is both efficient and accurate.

ViTTMs processes images using two token streams, one stream is compute-heavy but processes fewer tokens, while the other contains a larger number of tokens. Our intuition is that we can learn to exchange information between the two streams using *read-write heads*. The information in the compute-heavy stream undergoes more processing, thus we call them *process* tokens. Likewise, following NTM, we expect the stream with more tokens to be used to recall and store information, so we call them *memory* tokens.

This approach distinguishes ViTTMs from previous two-stream architectures like CrossViT and ViT-CoMer, as ViTTM concentrates substantial processing in a single stream. And, unlike Reversible ViTs, which reduce activation caching during training, ViTTM is specifically designed for computational efficiency during inference. Furthermore, ViTTMs offer advantages over token sparsification techniques. By processing fewer tokens while storing their information in memory, ViTTMs maintain both accuracy and efficiency. Unlike NTMs, which leverage memory for sequential tasks, our work applies memory to ViTs for
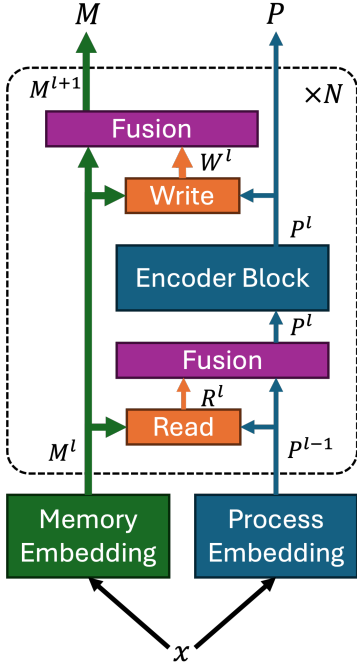
Figure 3. ViTTM Architecture. The ViTTM architecture is a NTM-ViT hybrid. In particular, ViTTM creates two views (or streams) of an input image, x, using two patch embedding layers. The memory stream, M, is created by a memory embedding layer, whereas the process stream, P, is created with a process embedding layer. Choose the memory stream to contain a greater number of tokens than the process *i.e.* $T > K$. The process and memory streams exchange information using *read* and *write* layers, followed by a fusion operation.

a non-sequential task, image recognition. Fig. 2 shows a comparison of NTMs to ViTTMs.

Comparison to LookupViT (ECCV 2024): The contemporaneous LookupViT [21], is also guided the observation that *processing fewer tokens through transformer blocks makes inference faster*. Similar to our approach, they exploit this fact by processing two sets of tokens, with the fewer tokens being relegated to the compute-heavy stream.

However, our effort differs from LookupViT, both conceptually and in implementation. LookupViT primarily utilizes cross attention, while we study various read-write and fusion mechanisms (Sec. 4.2) following prior work in NTMs. Furthermore, in contrast with LookupViT, we do not refine memory tokens. This follows NTM design, where memory is solely *read from* or *written to* at any time. Our decisions expand the LookupViT Pareto front by creating a model that is faster while being more accurate (Fig. 1).

3.2. ViTTM Architecture

Fig. 3 provides an overview of the ViTTM architecture. The ViTTM architecture is synthesis of the ViT and NTM architectures. Like ViT we convert an input image, $x \in \mathbb{R}^{3 \times H \times W}$, into non-overlapping patches. But unlike ViT, we create two sets of patches, not one, and convert each set into tokens. These patches are converted using embedding blocks into T *memory* tokens and K *process* tokens of dimension d.

We define the memory and process tokens (streams) as $M \in \mathbb{R}^{T \times d}$, and $P \in \mathbb{R}^{K \times d}$, respectively. After embedding, the tokens are processed using N repeating blocks. We index each block with an index l. Each of these ViTTM blocks, performs three operations in sequence: (1) *read from memory*, (2) *processing*, and (3) *write to memory*. The read and write operations return "read tokens" and "write tokens". We define the read and write tokens as $R \in \mathbb{R}^{K \times d}$ and $W \in \mathbb{R}^{T \times d}$, respectively. To ensure efficiency, we depart from ViT's approach of processing all tokens and instead apply the *processing* step only to the process tokens. The sequence is formally defined as:

$$R^l = \text{Read}(P^{l-1}, M^l) \tag{1}$$

$$P^l = \text{Encoder}(\text{Fusion}(P^{l-1}, R^l)) \tag{2}$$

$$W^l = \text{Write}(P^l, M^l) \tag{3}$$

$$M^{l+1} = \text{Fusion}(M^l, W^l) \tag{4}$$

The read and write operations are performed using *read-write* heads. Prior to the *processing* operation the read tokens are merged into the process tokens using a *fusion* operation, and a processing operation is carried out by the encoder block. After the *processing* operation the write tokens are merged into memory tokens.

The goal of ViTTMs is to be efficient while preserving accuracy. From Fig. 3 it is evident that the number of process tokens and the efficiency of the read-write heads will influence latency. This is because the encoder block's latency scales with the number of process tokens (Sec. 2.2), and repeated read/write can negate the benefits of reducing tokens. In addition, the effective use of memory impacts the task performance. Thus, for ViTTMs to be both efficient and effectively utilize memory, four core design decisions must be addressed:

1. How to initialize memory and process tokens?

2. How should tokens be selected for read/write?

3. How will the selected tokens be fused into the process/memory tokens?

4. Should memory tokens undergo processing?

Next we present the most effective choices for these design decisions, as identified by our ablations (Sec. 4.2).

3.2.1 Initializing Process and Memory Tokens

Our tokenization process for memory tokens follows the original ViT architecture [8]. The process is two steps:

First, we convert an image, $x \in \mathbb{R}^{3 \times H \times W}$ into a set of non-overlapping patches of size p. Second, we apply a learned linear projection, $f : \mathbb{R}^{3 \times H \times W} \mapsto \mathbb{R}^{\frac{H}{p} \times \frac{W}{p} \times d}$, to tokens of length d. The intuition is that the memory tokens should contain a rich representation of the image. However, the *process* tokens requires further consideration, because it is not obvious how they should be created.

We explore three options for initializing the *process* tokens: (1) *Latent* – learning a set of latent tokens; (2) *Downsample* – down-sampling the memory tokens to create the process tokens; and (3) *Patch* – learning a projection for the process tokens. In practice, they have identical performance. We opt to **learn separate projections for *memory* and *process* tokens**, since this is most similar ViT embeddings. To maintain ViTTM's efficiency, we ensure that there are fewer process tokens than memory tokens. This is achieved by using a larger patch size for process tokens.

3.2.2 Read-Write Heads

Read-write heads play a crucial role in selecting relevant information to read from or write to memory. A key consideration is the efficiency of reads and writes. To this end, we evaluate several designs for read-write heads: TokenSummary [31], Cross Attention [38], Latent Attention [7], and Linear Attention [19].

In this work, we adopt **Linear Attention**, as introduced by Katharopoulos *et al.* [19], to implement our read-write head. We justify our use of Linear Attention based on theoretical analysis and empirical results from our ablation studies (Sec. 4.2). It has the following benefits:

- The computational complexity of Linear Attention depends on the length of *one* input sequence, whereas Cross Attention depends on the length of *both* sequences.
- Linear attention is independent of sequence length, whereas TokenSummary is not. As a result, off-the-shelf token sparsification methods such as ToMe [1] can be used along side ViTTM. Additionally, we can use training methods such as MAE [17].
- We find that *Linear attention* has similar accuracy to cross-attention, while being more efficient.

Our read-write head computes linear attention between two input sequences $X_1 \in \mathbb{R}^{T \times d}$ and $X_2 \in \mathbb{R}^{K \times d}$. We compute the query, key, and value matrices using the weight matrices $W_q \in \mathbb{R}^{d \times c}$, $W_k \in \mathbb{R}^{d \times c}$, and $W_v \in \mathbb{R}^{d \times d}$, where d represents the *embedding dimension* and c represents the *latent embedding dimension*. The query, key, and value matrices are calculated as follows:

$$Q = X_2 W_q \in \mathbb{R}^{K \times c} \tag{5}$$

$$K = X_1 W_k \in \mathbb{R}^{T \times c} \tag{6}$$

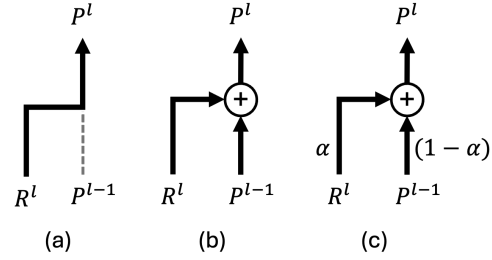$$V = X_1 W_v \in \mathbb{R}^{T \times d} \tag{7}$$

Figure 4. Illustration of our fusion implementations. (a) Erase (b) Add (c) Add-Erase. α is computed according to Eq. (10). Depending on the location of the fusion operation the inputs vary (Fig. 3).

Following Katharopoulos *et al.*, we compute the output sequence, V', as follows:

$$V' = \phi(Q) \left(\phi(K)^T V \right) \tag{8}$$

$$\phi(x) = 1 + \text{elu}(x) \tag{9}$$

During the *read* operation, we set $X_2 = P^{l-1}$, $X_1 = M^l$, and $V' = R^l$. Conversely, during the *write* operation, we set $X_2 = M^l$, $X_1 = P^l$, and $V' = W^l$. In our evalutation we use $c = d/4$, based on our ablations (Tab. 2).

3.2.3 Process-Memory Fusion

The *fusion* operation combines the read tokens R^l and write tokens W^l, from the *read-write heads* into the process and memory streams, respectively. We explore three implementations of fusion: *erase*, *add*, and *add-erase* fusion (Fig. 4).

Erase replaces the previous process tokens (P^{l-1}) with the read tokens (R^l), i.e. erasing the prior process tokens. *Add* sums the read tokens (R^l) with the process tokens (P^{l-1}). *Add-Erase* creates the new process tokens P^l as a convex combination of the read tokens R^l and the previous process tokens P^{l-1}, using combination weights α and $1 - \alpha$. The combination weights α are computed according to Equation Eq. (10) (bias omitted). We project the average read tokens, $R_{avg.} \in \mathbb{R}^d$, using the weight matrix $W_\alpha \in \mathbb{R}^{K \times d}$ and apply the sigmoid non-linearity σ.

$$\alpha = \sigma(W_\alpha R^l_{avg.}) \in \mathbb{R}^K \tag{10}$$

In our evaluation we adopt **Add** fusion because this has the best performance with **Linear Attention** (Tab. 4).

3.2.4 Processing in the Memory Stream

Whether the memory tokens should undergo processing is a key consideration. Processing allows for the further enrichment of memory tokens, while trading off efficiency. We explore the effect of adding a multi-layer preceptron (MLP) in the memory stream with varying bottleneck dimensions in Tab. 4. Unlike LookupViT [21], the benefit of an MLP

is marginal and we do not process the memory stream explicitly. We find that not refining the memory stream can make training unstable — normalization helps, but increasing batch-size is more effective.

4. Evaluation

We evaluate ViTTM design decisions and compare with existing work. Sec. 4.1 describes our experimental setup, and Sec. 4.2 explores design decisions of ViTTM's. We compare with other methods on classification and segmentation in Sec. 4.3.

4.1. Experimental Setup

Training: We follow ViT training recipes laid out in [34]. Specifically, we pre-train our models on ImageNet-21K [30], and subsequently fine-tune them on ImageNet-1K [6] at 224px image resolution. For semantic segmentation, we initialize ViTTM backbone models with fine-tuned ImageNet-1K weights. We compare against ViT-B/16 backbones fine-tuned at 224px for fair comparison. Our training recipes are included in the Appendix.

Measurement: FLOP measurements were performed with the *fvcore* library [10], and latency was measured using the PyTorch *benchmark* module [26] with a batch size of 256. Segmentation experiments and benchmarking were performed using the MMSegmentation library [5]. Benchmark and evaluations were performed on both a single 80GB A100 GPU and a 24GB A30 GPU.

Notation: We denote our ViTTM configurations as follows: ViTTM-B$_{(49,64)}$ is a "ViTTM-B model with 49 process tokens and 64 memory tokens". For models *without* memory tokens (*i.e.* baseline models), we use ϕ to denote the absence of memory tokens.

For reproducibility, our code is open source [18].

4.2. Ablations

In this section, we perform ablation studies on the four core design decisions of ViTTMs (Sec. 3.2). We conduct ablations to answer the following:

1. How to initialize memory and process tokens? (Sec. 4.2.1 and Sec. 4.2.4)
2. How tokens will be selected for read/write? (Sec. 4.2.2 and Sec. 4.2.3)
3. How will the selected tokens be fused into the process/memory tokens? (Sec. 4.2.2)
4. Should memory tokens undergo processing? (Sec. 4.2.5)

To assess the efficacy of each design choice, we fine-tune and evaluate on ImageNet-1K. We allocate a fixed amount training time, 100 epochs, for each ablation. The ablations in Sec. 4.2.1 and Sec. 4.2.2 are conducted using a ViTTM-S model, whereas those in Sec. 4.2.3, Sec. 4.2.4, and Sec. 4.2.5 ablations use a ViTTM-B model.

4.2.1 Memory and Process Initialization

We evaluate the initialization method for process tokens as defined in Sec. 3.2.1. Ultimately, the choice of initialization scheme has no impact on accuracy. Therefore, we opted for *Patch* initalization for all of our models since it is the standard method for ViT architectures.

4.2.2 Read-Write Head and Fusion Choice

We explore combinations of read-write heads and fusion mechanisms outlined in Sec. 3.2. Tab. 1 displays the ablation results for read-write head and fusion mechanisms. We observe that all read-write heads provide comparable accuracy, primarily differing in computational cost. As stated in Sec. 3.2.2, we use Linear Attention [19] due to its efficiency and comparable accuracy to Cross Attention.

4.2.3 Effect of Read-Write Head Latent Dimension

Tab. 2 illustrates the impact of varying the latent embedding dimension c. Reducing the latent embedding dimension relative to the embedding dimension, d, does not reduce accuracy, but can provide substantial latency benefits. As shown in Tab. 2, setting $c = d/4$ reduces latency by 100ms with negligible impact on Top-1 accuracy. Given this result, we choose $c = d/4$ for all evaluations in this work.

4.2.4 Effect of Memory and Process Token Number

In Tab. 3, we present how the number of memory and process tokens affects both the computational cost and accu-

| | Read-Write Head | | | |
Fusion	Token Summary	Cross Attn.	Latent Attn.	Linear Attn.
Erase	72.49	3.94	0.33	2.43
Add	75.86	76.26	**76.17**	**76.26**
Add-Erase	**76.26**	**76.33**	75.82	74.74
GFLOPs	1.86	2.92	3.07	2.62

Table 1. Ablation over read-write head and fusion mechanisms. Each cell represents the Top-1 ImageNet1K accuracy for a read-write head and fusion mechanism combination. The GFLOPs are measured for each read-write head using *Add* fusion (fusion contributions to GFLOPs negligible). TokenSummary has the lowest computational cost, followed by Linear Attention, Cross Attention, and finally Latent Attention. *Add* fusion works best, whereas *Add-Erase* fusion degrades the performance of Latent and Linear Attention. *Erase* fusion performs the worst for the attention-based heads, and best when used with Token Summary.

c	GFLOPs (M)$\downarrow$	Latency (ms)$\downarrow$	Top-1 (%)$\uparrow$
$d/4$	7.10	250.7	78.4
$d/2$	8.04	288.9	78.0
d	9.92	352.2	78.3

Table 2. Ablation over the *latent embedding dimension*, c for the read-write head on ViTTM-B$_{(32,16)}$ ($d = 768$). Accuracy is computed using the memory tokens rather than the process tokens. We use $\uparrow$/$\downarrow$ notation to indicate *"larger/smaller is better"*. We find latency reductions saturate when $c < d/4$.

racy of ViTTM-B. We find: (1) *memory tokens improve accuracy*, (2) *the marginal accuracy increase of additional memory tokens is small*, and (3) *the marginal accuracy increase of additional process tokens is large*.

First, it is clear that adding memory tokens improves accuracy. When comparing ViT-B$_{(49,\phi)}$ to ViTTM-B$_{(49,64)}$, there is a 7.2% improvement in Top-1 accuracy. Second, the marginal benefit of extra memory tokens is minimal. This is evident when we compare ViTTM-B$_{(49,64)}$, ViTTM-B$_{(49,196)}$, and ViTTM-B$_{(49,256)}$, where adding extra memory tokens yield minimal accuracy gains. Third, the marginal benefit of extra process tokens is high. As seen when comparing ViTTM-B$_{(49,64)}$ to ViTTM-B$_{(64,64)}$. The addition of process tokens, while keeping memory tokens constant, increases accuracy by 1.2%.

4.2.5 Effect of Memory Stream Non-linearity

LookupViT proposed using a non-linearity to their lookup tokens (analogous to our memory stream) for accuracy. We

Process Tokens	Memory Tokens	GFLOPs$\downarrow$	Latency $\downarrow$ (ms)	Top-1 $\uparrow$ (%)
49	ϕ	4.37	138.3	72.2
196	ϕ	16.87	527.4	81.0
16	64	3.39	125.0	70.4
16	196	6.37	248.0	70.9
16	256	7.72	298.9	71.2
49	64	6.94	233.9	79.4
49	196	9.92	357.7	79.3
49	256	11.27	408.7	79.8
64	64	8.56	285.3	80.6
64	196	11.53	409.0	80.4
64	256	12.88	460.0	80.9

Table 3. Ablation of number of tokens for the process and memory streams on ViTTM-B; we use the ϕ to indicate models *without* memory tokens *i.e.* baseline models. Although memory tokens improve accuracy, the marginal improvement is small, *e.g.*, 64 → 256 memory tokens. Increasing the number of process tokens has the greatest impact on accuracy, *e.g.*, 49 → 64 process tokens.

report that this is inefficient (Tab. 4). An MLP increases FLOPS by 14%−56% and latency by 13%−47%, while having negligible effects on accuracy.

4.3. Comparison with other work

We evaluate our ViTTM models on ImageNet-1K classification (Sec. 4.3.1) and ADE20K semantic segmentation (Sec. 4.3.2) using Segmenter [35] from the MMSegmentation library [5]. For image classification, we compare against existing two-stream architectures, and use ViT and DeiT models as baselines [8, 36]. For semantic segmentation, we compare with a ViT-B/16 model from the timm library [40]. Based on our ablations we configure our ViTTM model with **Linear Attention** read-write heads ($c = d/4$), **Add** fusion, and with **no** memory stream non-linearity.

4.3.1 Image Classification (ImageNet-1K)

ViTTM offers competitive accuracy-latency trade-offs on ImageNet-1K. Tab. 5 presents our evaluation with existing methods. We achieve lower latency while matching or nearly matching accuracy of other methods, thereby expanding the Pareto frontier (Fig. 1). For example, compared with the similar latency LookupViT$_{3\times3}$, ViTTM-B$_{(28,28)}$ has 5.5% greater accuracy. When compared with the similar accuracy LookupViT$_{7\times7}$, our ViTTM-B$_{(28,28)}$ has a 145.1ms (38%) lower latency.

4.3.2 Semantic Segmentation (ADE20K)

ViTTM also achieves competitive results on ADE20K across multiple devices, as illustrated in Tab. 6. Our ViTTM-B$_{(28,28)}$ matches ViT-B/16$_{224}$ on ADE20K semantic segmentation while improving FPS by 37% (23.8 FPS to 32.5 FPS) on an A100 GPU, and by 94% (13.8 FPS to 26.8 FPS) on an A30 GPU. In this work, we do not investigate the reason for the discrepancy between devices.

Mem. Stream Block	Params (M)	GFLOPs$\downarrow$	Latency $\downarrow$ (ms)	Top-1 $\uparrow$ (%)
None	37.9	9.92	358.2	78.2
MLP ($r = 0.5$)	39.7	11.31	403.6	78.3
MLP ($r = 1.0$)	41.5	12.70	444.9	78.3
MLP ($r = 2.0$)	45.0	15.47	525.1	78.2

Table 4. Ablation over the choice of non-linearity for the memory stream on ViTTM-B. r is the scaling ratio applied to the embedding dimension d, thus the hidden dimension of the MLP is $r \times d$. The inclusion of an MLP layer seems to have no effect on accuracy while increasing the FLOPs and latency of our model.

Model Class	Model	Params (M)	GFLOPs ↓	Latency (ms)↓	Top-1(%)↑
ViT/DeiT	ViT-B/32 [40]	88	4.37	138.3	72.2
	ViT-B/16 [40]	87	16.87	529.5	81.0
	DeiT-B/16 [36]	87	16.87	529.7	81.8
Two-Stream	CrossViT-B [3]	105	21.22	728.1	82.2
	CrossViT-18 [3]	43	9.05	374.1	82.5
	CrossViT-18† [3]	44	9.50	378.2	82.8
	Rev-ViT-B [24]	86	17.49	556.5	81.8
	LookupViT$_{3\times3}$ [21]	90	5.26	230.5	77.9
	LookupViT$_{5\times5}$ [21]	90	6.94	297.2	81.6
	LookupViT$_{7\times7}$ [21]	90	9.45	379.5	83.0
	LookupViT$_{10\times10}$ [21]	90	14.80	563.4	83.9
Ours	ViTTM-B$_{(28,28)}$	127	7.08	234.1	82.9
	ViTTM-B$_{(32,16)}$	125	7.10	251.5	80.9

Table 5. Comparison of ViTTM with state-of-the-art methods on image classification (ImageNet-1K). ViTTMs are much faster (lower latency) than state-of-the-art methods while matching their accuracy is most cases. Fig. 1 depicts the data in this table. Latency was measured on a 80GB A100 with batch size 256. *Notes*: The ViT baseline model is the 224 resolution fine-tuned model from [8], available from `timm` [40]. LookupViT does not have a public implementation, as such we implement a version following the paper.

Model (M)	FPS (A100) ↑	FPS (A30) ↑	mIoU ↑
ViT-B/16$_{384}$ [35]	23.8	13.8	48.06
ViT-B/16$_{224}$	23.8	13.8	45.65
ViTTM-B$_{(28,28)}$	*(+37%)* 32.5	*(+94%)* 26.8	45.17
ViTTM-B$_{(32,16)}$	*(+38%)* 32.8	*(+94%)* 26.7	43.60

Table 6. Results for semantic segmentation (ADE20K) using the Segmenter [35] segmentation method. FPS measurements are taken on 80GB NVIDIA A100 and 24GB A30 GPUs.

5. Discussion and Future Work

Two-Stream Architectures: It is evident from both prior work and our experiments that two-stream architectures can be used to create both efficient and accurate vision models, *e.g.*, LookupViT, CrossViT, and ViTTMs. However, two-stream architectures present more design choices than ViTs, leaving much to explore about what constitutes an optimal design. For example, although we ablate many design choices (Sec. 4.2), we do not consider differing embedding dimensions per stream, varying depth, or the effect of self-supervised learning. Previous studies [3] have demonstrated the effectiveness of altering embedding dimensions, yet its interplay with other parameters remains unclear. Changing the embedding dimension has been shown to be effective in prior work [3], but how it is related to other parameters is unknown. Moreover, we also find that two-stream architectures can suffer from training instabilities (Sec. 3.2.4). An important direction for future work is the rigorous study of both architectural choices and the training dynamics of two-stream architectures.

Read-Write Heads: An important consideration in our design is the choice of read-write heads. As stated in Sec. 3.2.2, read-write heads must be efficient. We believe that efficient implementation of read-write mechanisms is an important direction for future work. Actually obtaining latency reductions from FLOP reductions is important for the further development and use of alternative architectures.

Uses of Memory Stream for Multi-Modal application: In this work we ablate how the process stream is be initialized, *i.e.* how it is populated with tokens (Sec. 3.2.1). However, we do not explore how the memory tokens should be initialized. It is possible to initialize the memory tokens using another modality, thus creating a multi-modal model. Such a model would read/write (fuse) information across modalities. Recent work, in generative models use two-stream architectures for this purpose [9, 13].

6. Conclusion

We introduced ViTTM, a ViT-NTM hybrid, that integrates memory into vision transformers to create an efficient and accurate vision model. Evaluation on ImageNet1K and ADE20K demonstrated that ViTTM accuracy matches or exceeds prior work. ViTTM-B is 56% faster (2.4× fewer FLOPs) than ViT-B while having an accuracy of 82.9% (+1.9%) on ImageNet1K. On ADE20K, ViTTM-B achieves 37%-94% higher FPS than ViT-B. Our current work illustrated the effectiveness of ViTTMs, and we anticipate that their efficiency can be improved and extended to other tasks. Future work that explores latency optimization of the read-write mechanisms, such as fusing operators, would be an effective extension.

References

[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster, 2023. 1, 3, 5, 11, 12

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 2

[3] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification, 2021. 2, 8

[4] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9640–9649, 2021. 2

[5] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 6, 7

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[7] Rares Dolga, Marius Cobzarenco, and David Barber. Latent attention for linear time transformers. *arXiv preprint arXiv:2402.17512*, 2024. 5

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *arXiv*, 2020. 1, 2, 4, 7, 8, 12

[9] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024. 8

[10] FAIR. fvcore. https://github.com/facebookresearch/fvcore, 2024. 6

[11] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6824–6835, October 2021. 2

[12] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive token sampling for efficient vision transformers. In *European Conference on Computer Vision*, pages 396–414. Springer, 2022. 3

[13] Zhengcong Fei, Mingyuan Fan, Changqian Yu, and Junshi Huang. Flux that plays music. *arXiv preprint arXiv:2409.00587*, 2024. 8

[14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. 1, 3

[15] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. 3

[16] Joakim Bruslund Haurum, Sergio Escalera, Graham W. Taylor, and Thomas B. Moeslund. Which tokens to use? investigating token reduction in vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 773–783, October 2023. 3

[17] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2022. 2, 5

[18] Purvish Jajal. EfficientTTMs. https://github.com/pjjajal/EfficientTTMs, 2024. 6

[19] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020. 5, 6

[20] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 784–794, 2022. 3

[21] Rajat Koner, Gagan Jain, Prateek Jain, Volker Tresp, and Sujoy Paul. Lookupvit: Compressing visual information to a limited number of tokens. *arXiv*, 2024. 1, 2, 4, 5, 8

[22] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations, 2022. 3

[23] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 2

[24] Karttikeya Mangalam, Haoqi Fan, Yanghao Li, Chao-Yuan Wu, Bo Xiong, Christoph Feichtenhofer, and Jitendra Malik. Reversible vision transformers, 2023. 2, 8

[25] Muzammal Naseer, Kanchana Ranasinghe, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. *arXiv*, 2021. 3

[26] PyTorch Foundation. PyTorch Benchmark: PyTorch Tutorials Documentation. url: https://pytorch.org/tutorials/recipes/recipes/benchmark.html, 2024. 6

[27] Jack W Rae, Jonathan J Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes, 2016. 1, 3

[28] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification, 2021. 1, 3

[29] Cedric Renggli, André Susano Pinto, Neil Houlsby, Basil Mustafa, Joan Puigcerver, and Carlos Riquelme. Learning to merge tokens in vision transformers. *arXiv preprint arXiv:2202.12015*, 2022. 3

[30] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses, 2021. 6

[31] Michael S. Ryoo, Keerthana Gopalakrishnan, Kumara Kahatapitiya, Ted Xiao, Kanishka Rao, Austin Stone, Yao Lu, Julian Ibarz, and Anurag Arnab. Token turing machines. In *CVPR*, 2023. 1, 3, 5

[32] Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? *arXiv*, 2022. 3

[33] Mark Sandler, Andrey Zhmoginov, Max Vladymyrov, and Andrew Jackson. Fine-tuning image transformers using learnable memory. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12155–12164, 2022. 1, 3

[34] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers, 2022. 2, 6

[35] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7262–7272, 2021. 7, 8

[36] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357, 2021. 2, 7, 8

[37] Hugo Touvron, Matthieu Cord, and Hervé Jégou. DeiT III: Revenge of the ViT, 2022. 2

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *arXiv*, 2017. 2, 5

[39] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks, 2015. 1, 3

[40] Ross Wightman. PyTorch Image Models (TIMM). https://github.com/rwightman/pytorch-image-models, 2019. 7, 8, 12

[41] Qingyang Wu, Zhenzhong Lan, Kun Qian, Jing Gu, Alborz Geramifard, and Zhou Yu. Memformer: A memory-augmented transformer for sequence modeling. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang, editors, *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, pages 308–318, Online only, Nov. 2022. Association for Computational Linguistics. 3

[42] Chunlong Xia, Xinliang Wang, Feng Lv, Xin Hao, and Yifeng Shi. Vit-comer: Vision transformer with convolutional multi-scale feature interaction for dense predictions, 2024. 2

[43] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10809–10818, 2022. 3