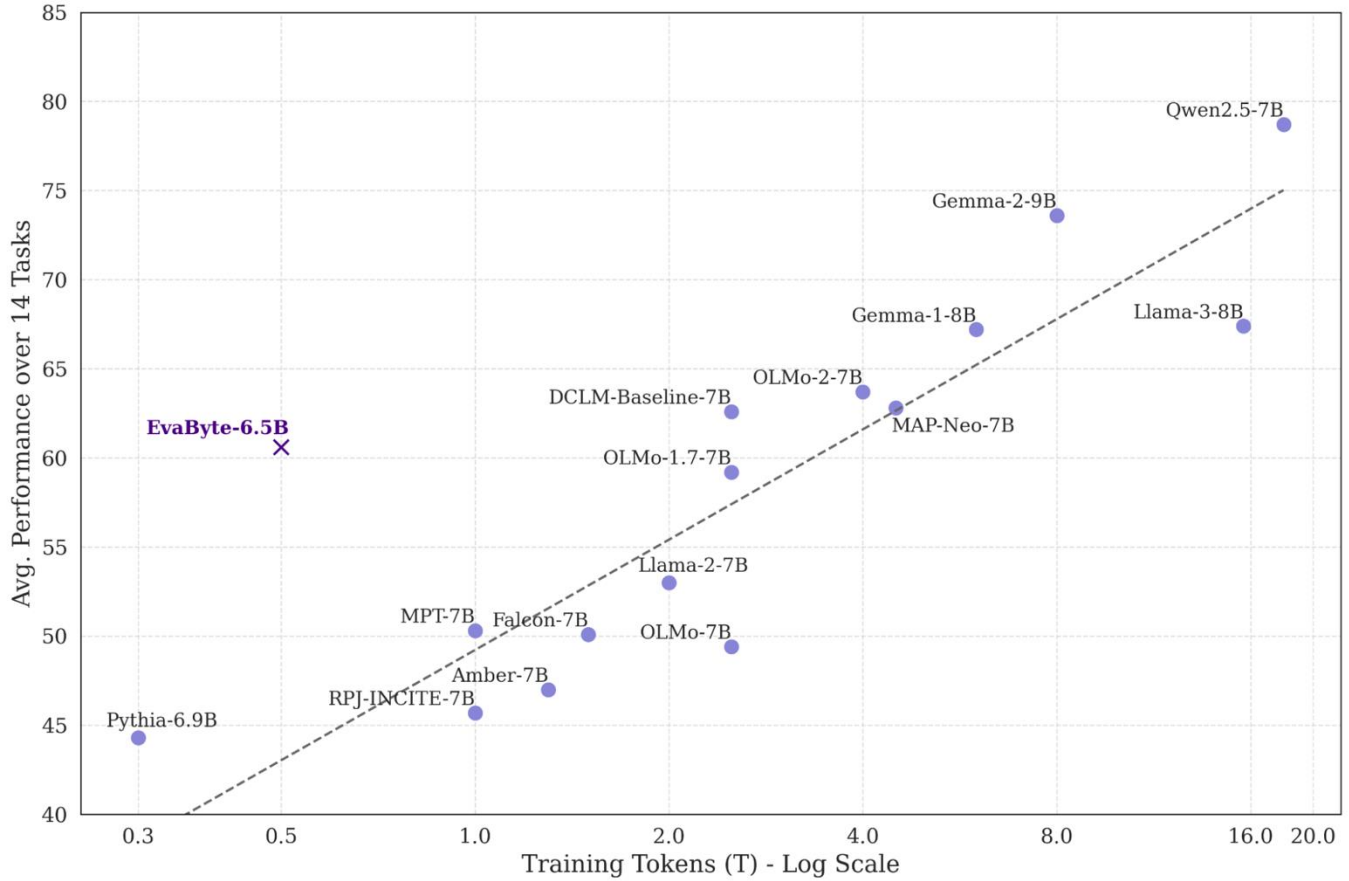
EvaByte: Efficient Byte-level Language Models at Scale

Lin Zheng, Xueliang Zhao, Guangtao Wang, Chen Wu, David Dong, Angela Wang, Mingran Wang, Yun Du, Haige Bo, Amol Sharma, Bo Li, Kejie Zhang, Changran Hu, Urmish Thakker, Lingpeng Kong

Blog post: https://hkunlp.github.io/blog/2025/evabyte
Huggingface: https://huggingface.co/EvaByte
Codebase: https://github.com/OpenEvaByte/evabyte

EvaByte

- Highly competitive perf. with 5x less training data



Why Avoid Tokenization?

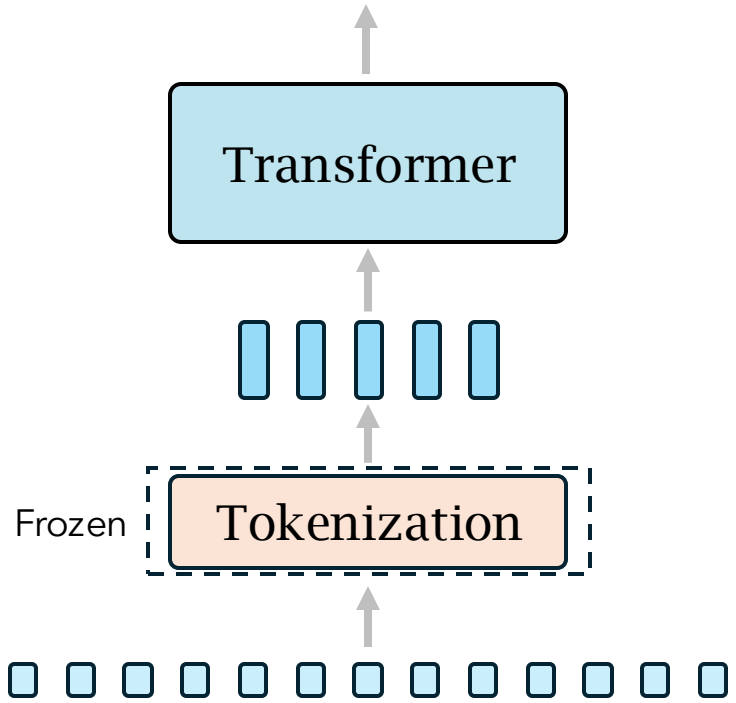
- Tokenizers are externally built and detached from training
- Make LMs not end-to-end and fragile
  - Glitch tokens, prompt boundary issues, …

User:
What's wrong with the code below?

```
def convert_name(name):
    bool ForCanBeConverted = name == ".pt"
    return ForCanBeConverted
```

Qwen2.5-7B-Instruct:
There are a few issues with the provided code snippet:

1. The variable name `bool<|im_end|>



Challenges

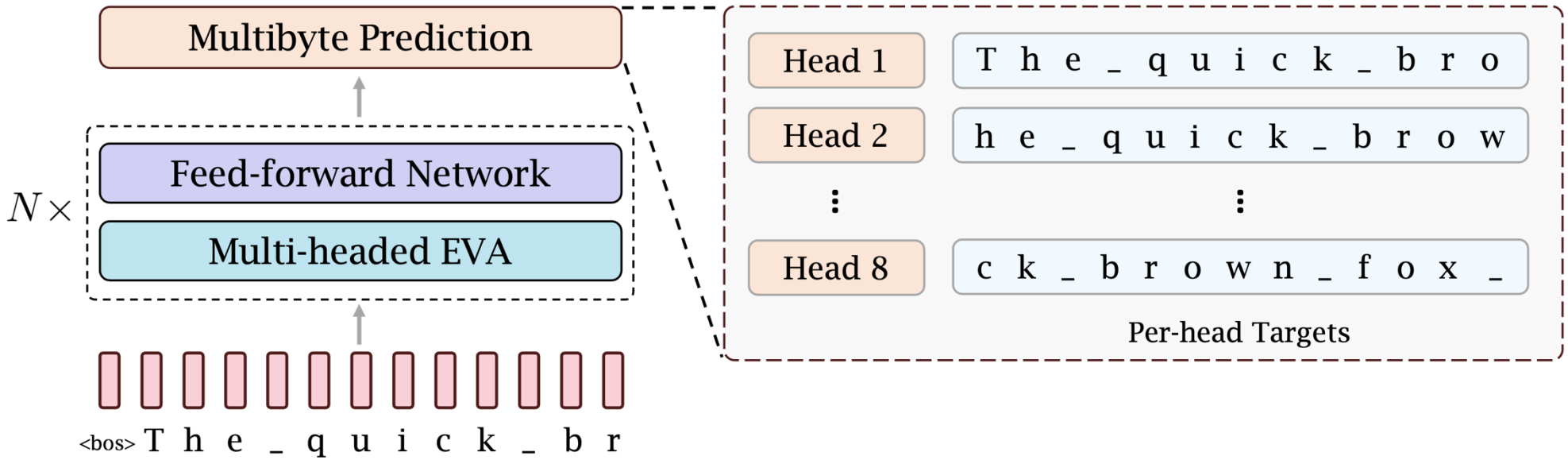
- Byte sequences are much **longer** than tokenized counterparts
  - ~4x longer in our text data mix
  - >64x longer for images, audio, …
- Much higher training and inference cost

3.9 Bytes ≈ 1 Token

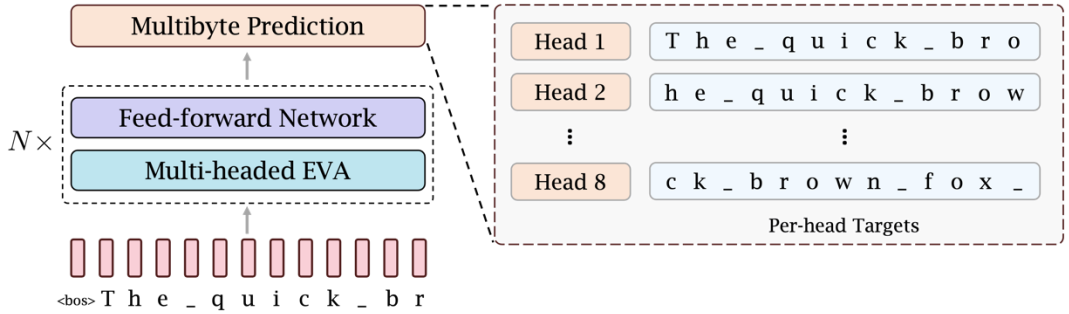
```
def greet(name):
    """Function to greet a user by their name."""“
    if name:
        print(f"Hello, {name}!")
    else:
        print("Hello, World!")
```

Approach

- Efficient byte modeling with streamlined arch-side changes
  - Multibyte Prediction
  - Efficient attention with EVA



Multibyte Prediction



- Next several bytes are predicted jointly instead of one
  - Losses from different heads are simply summed together

$$\ell = -\sum_{n=0}^{N-1}\sum_{k=1}^{K} \log p_{\boldsymbol{\theta}}\left(\mathbf{b}_{n+k} \mid \mathbf{b}_{1:n}\right)$$

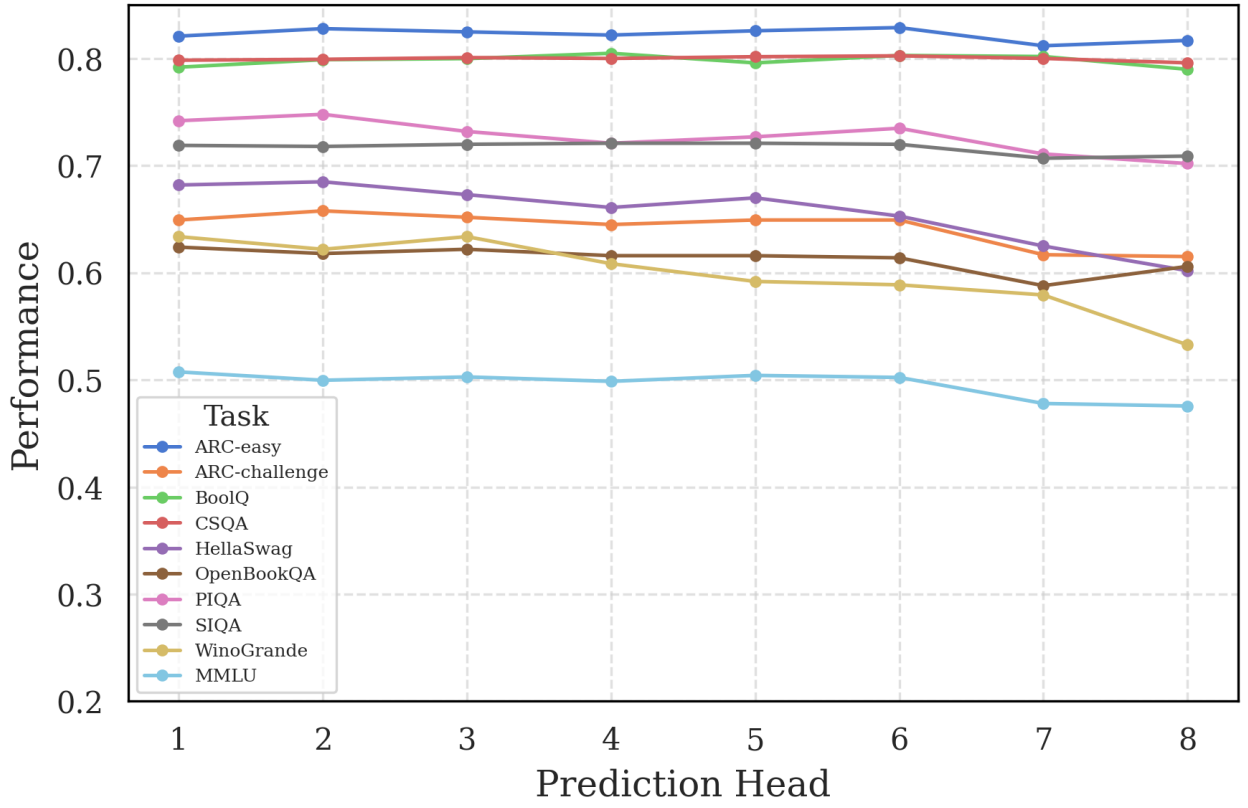
  - Almost no training overhead with 8 heads due to small vocab size (320)

- We tried power-decayed weighting but observed no perf. diff.

[1] Stern, Mitchell, et al. Blockwise parallel decoding for deep autoregressive models. NeuIPS 2018.
[2] Qi, Weizhen, et al. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. EMNLP 2020.
[3] Cai, Tianle, et al. Medusa: Simple llm inference acceleration framework with multiple decoding heads. ICML 2024.
[4] Gloeckle, Fabian, et al. Better & faster large language models via multi-token prediction. ICML 2024.

Multibyte Prediction

- All prediction heads are learned quite effectively
- Downstream choice-task performance stable across heads



Efficient Attention

- Multibyte pred. alone is not enough to speed up byte models
  - Attention becomes the bottleneck as the context fills up rapidly with byte streams

$$\mathrm{Attn}\left(\mathbf{q}_n, \{\mathbf{k}_m\}, \{\mathbf{v}_m\}\right) = \sum_m \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right)}{\sum_{m'} \exp(\mathbf{q}_n^\top \mathbf{k}_{m'})} \mathbf{v}_m^\top$$

- Solution: EVA -- a simple yet efficient approx. to attention
  - A simple variant of linearized attention

[1] Zheng, Lin, et al. "Efficient attention via control variates." ICLR 2023 Oral.

Revisiting Linearized Attention

- Standard attention runs in quadratic runtime

$$\mathrm{Attn}\left(\mathbf{q}_n, \{\mathbf{k}_m\}, \{\mathbf{v}_m\}\right) = \sum_m \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right)}{\sum_{m'} \exp(\mathbf{q}_n^\top \mathbf{k}_{m'})} \mathbf{v}_m^\top$$

- Approximation: linearizing the exponential dot product

$$\exp(\mathbf{x}^\top \mathbf{y}) \approx \phi\left(\mathbf{x}\right)^\top \phi\left(\mathbf{y}\right)$$

$$\sum_m \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{m'} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)} \approx \sum_m \frac{\phi\left(\mathbf{q}_n\right)^\top \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{m'} \phi\left(\mathbf{q}_n\right)^\top \phi\left(\mathbf{k}_{m'}\right)} = \frac{\phi\left(\mathbf{q}_n\right)^\top \sum_m \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\phi\left(\mathbf{q}_n\right)^\top \sum_{m'} \phi\left(\mathbf{k}_{m'}\right)}$$

- Observation: linearized attention compresses all KV information to a single global memory state

EVA

- EVA: global memory distributed into local slots

$$\sum_m \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{m'} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)} = \sum_{c=1}^{C} \frac{\sum_{m\in\mathcal{M}_c} \exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{c'=1}^{C} \sum_{m'\in\mathcal{M}_{c'}} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)}$$ Chunking

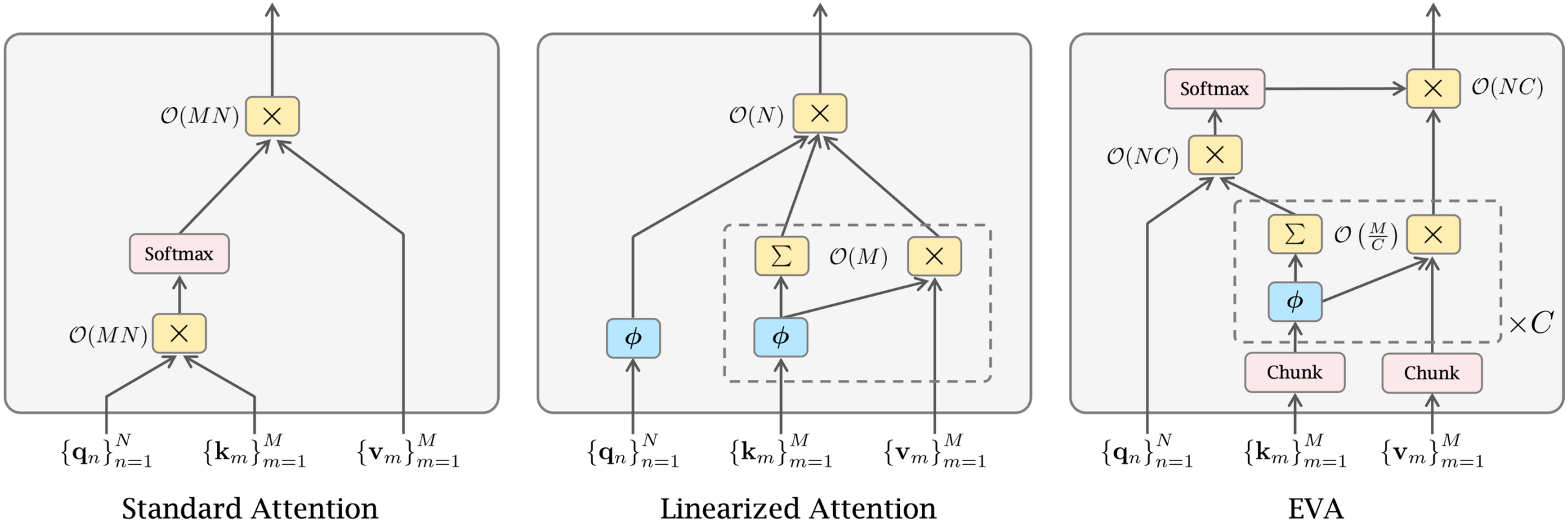
$$= \sum_{c=1}^{C} \frac{\sum_{m\in\mathcal{M}_c} \exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{c'=1}^{C} \sum_{m'\in\mathcal{M}_{c'}} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)} \frac{\sum_{m\in\mathcal{M}_c} \exp(\mathbf{q}_n^\top \mathbf{k}_m)}{\sum_{m'\in\mathcal{M}_c} \exp(\mathbf{q}_n^\top \mathbf{k}_{m'})}$$ Multiply-by-1

$$= \sum_{c=1}^{C} \underbrace{\frac{\sum_{m\in\mathcal{M}_c} \exp(\mathbf{q}_n^\top \mathbf{k}_m)}{\sum_{c'=1}^{C} \sum_{m'\in\mathcal{M}_{c'}} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)}}_{\text{“Attention score" for each chunk}} \underbrace{\frac{\sum_{m\in\mathcal{M}_c} \exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right) \mathbf{v}_m^\top}{\sum_{m'\in\mathcal{M}_c} \exp(\mathbf{q}_n^\top \mathbf{k}_{m'})}}_{\text{“Attention" per chunk}}$$ Rearranging

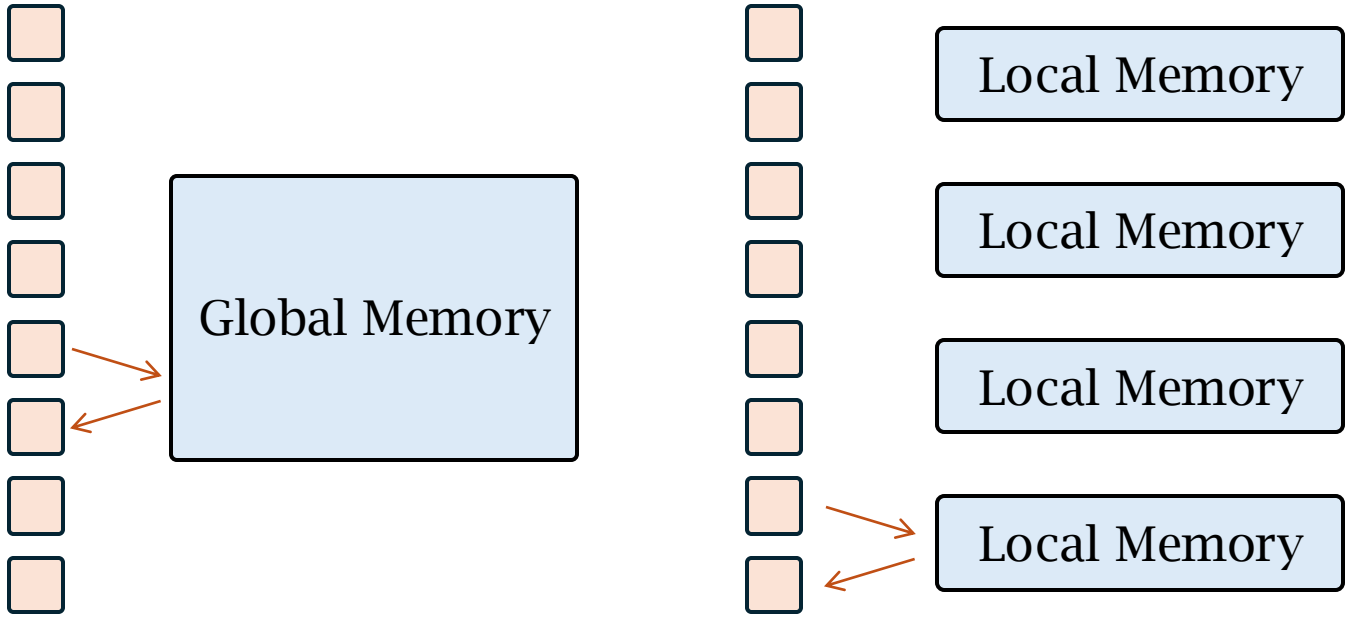
$$\approx \sum_{c=1}^{C} \frac{\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right)}{\sum_{c'=1}^{C} \psi\left(\mathbf{q}_n, \{\mathbf{k}'_m\}_{m'\in\mathcal{M}_{c'}}\right)} \frac{\phi\left(\mathbf{q}_n\right)^\top \sum_{m\in\mathcal{M}_c} \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\phi\left(\mathbf{q}_n\right)^\top \sum_{m\in\mathcal{M}_c} \phi\left(\mathbf{k}_m\right)}$$ Double approx.

EVA

- Interpolation between standard and linearized attention



EVA



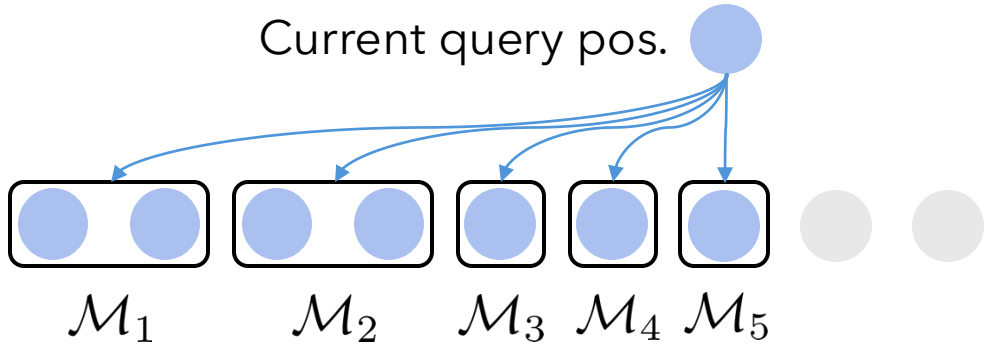
- Expands the design space of linearized attention
- Easier to improve local memory than global
- Complementary to research in better organizing memory states
  - Gated RFA, GLA, Gated DeltaNet, TTT, Titans, HGRNs, RWKVs, Mambas, …

[1] Peng, Hao, et al. "Random feature attention." ICLR 2021.
[2] Yang, Songlin, et al. "Gated linear attention transformers with hardware-efficient training." ICML 2024.
[3] Dao, Tri, and Albert Gu. "Transformers are ssms: Generalized models and efficient algorithms through structured state space duality." ICML 2024.
[4] Yang, Songlin et al. "Gated Delta Networks: Improving Mamba2 with Delta Rule." ICLR 2025.
[5] Yang, Songlin et al. "Parallelizing Linear Transformers with the Delta Rule over Sequence Length." NeurIPS 2024.
[6] Qin, Zhen et al. "Hierarchically Gated Recurrent Neural Network for Sequence Modeling." NeurIPS 2023.
[7] Sun, Yu et al. "Learning to (Learn at Test Time): RNNs with Expressive Hidden States." ArXiv: 2407.04620.
[8] Behrouz, Ali et al. "Titans: Learning to Memorize at Test Time." ArXiv: 2501.00663.
[9] Peng, Bo, et al. "Rwkv: Reinventing rnns for the transformer era." arXiv: 2305.13048.

EVA

$$\mathbf{EVA} = \sum_{c=1}^{C} \frac{\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right)}{\sum_{c'=1}^{C} \psi\left(\mathbf{q}_n, \{\mathbf{k}'_m\}_{m'\in\mathcal{M}_{c'}}\right)} \frac{\phi(\mathbf{q}_n)^\top \sum_{m\in M_c} \phi(\mathbf{k}_m)\, \mathbf{v}_m^\top}{\phi(\mathbf{q}_n)^\top \sum_{m\in M_c} \phi(\mathbf{k}_m)}$$

- **Chunking** decides approx. fidelity-efficiency trade off
- Hybrid design: fixed-size distant chunks + singleton neighbors



- Aka chunk-wise linearized attention + block attention
- Connection to Scatterbrain, Infini-attention, LoLCATs, LESS, NSA, …

[1] Chen, Beidi, et al. "Scatterbrain: Unifying sparse and low-rank attention." NeurIPS 2021.
[2] Munkhdalai, Tsendsuren et al. "Leave no context behind: Efficient infinite context transformers with infini-attention." arXiv:2404.07143.
[3] Zhang, Michael, et al. "LoLCATs: On Low-Rank Linearizing of Large Language Models." ICLR 2025.
[4] Dong, Harry, et al. "Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference." ICML 2024.
[5] Yuan, Jingyang, et al. "Native sparse attention: Hardware-aligned and natively trainable sparse attention." arXiv:2502.11089.

EVA

$$\mathbf{EVA} = \sum_{c=1}^{C} \frac{\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right)}{\sum_{c'=1}^{C} \psi\left(\mathbf{q}_n, \{\mathbf{k}'_m\}_{m'\in\mathcal{M}_{c'}}\right)} \frac{\phi\left(\mathbf{q}_n\right)^\top \sum_{m\in M_c} \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\phi\left(\mathbf{q}_n\right)^\top \sum_{m\in M_c} \phi\left(\mathbf{k}_m\right)}$$

- $\psi(\cdot)$ controls interpolation between linearized and standard attention

$$\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right) = \sum_{m\in\mathcal{M}_c} \exp(\mathbf{q}_n^\top \mathbf{k}_m)$$

Exact computation but quadratic

$$\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right) = \sum_{m\in\mathcal{M}_c} \widetilde{\phi}(\mathbf{q}_n)^\top \widetilde{\phi}(\mathbf{k}_m)$$

- **Double linearization** with $\phi(\cdot)$ and $\tilde{\phi}(\cdot)$
- Efficient but poor interpolation:
  - $C = 1$: linearized attention with $\phi(\cdot)$
  - $C = M$: linearized attention with $\tilde{\phi}(\cdot)$

$$\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m\in\mathcal{M}_c}\right) = \exp\left(\mathbf{q}_n^\top \sum_{m\in\mathcal{M}_c} \alpha_m \mathbf{k}_m\right),$$

$$\alpha_m = \exp\left(\boldsymbol{\omega}_\psi^\top \mathbf{k}_m\right) / \sum_{m'\in\mathcal{M}_c} \exp\left(\boldsymbol{\omega}_\psi^\top \mathbf{k}_{m'}\right)$$

- Heuristic used in EVA
- Efficient and **effective interpolation**:
  - $C = 1$: linearized attention with $\phi(\cdot)$
  - $C = M$: exact full attention
- We found this produced better perf.

EVA

$$\mathbf{EVA} = \sum_{c=1}^{C} \frac{\psi\left(\mathbf{q}_n, \{\mathbf{k}_m\}_{m \in \mathcal{M}_c}\right)}{\sum_{c'=1}^{C} \psi\left(\mathbf{q}_n, \{\mathbf{k}'_m\}_{m' \in \mathcal{M}_{c'}}\right)} \frac{\phi\left(\mathbf{q}_n\right)^\top \sum_{m \in M_c} \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\phi\left(\mathbf{q}_n\right)^\top \sum_{m \in M_c} \phi\left(\mathbf{k}_m\right)}$$

- Parameterizing $\phi(\cdot)$ as in Performer

$$\boldsymbol{\phi}(\mathbf{x}) = \left[\exp\left(\mathbf{x}^\top \boldsymbol{\omega}_1 - \frac{1}{2}\|\mathbf{x}\|^2\right), \ldots, \exp\left(\mathbf{x}^\top \boldsymbol{\omega}_S - \frac{1}{2}\|\mathbf{x}\|^2\right)\right] \in \mathbb{R}^S, \quad \boldsymbol{\omega}_s \sim \mathcal{N}(0, \mathbb{I})$$

- We use **scalar features** ($S = 1$) with degenerate form
  - Very similar perf. to $S > 1$ but simpler implementation

$$\frac{\cancel{\phi(\mathbf{q}_n)} \sum_{m \in M_c} \phi\left(\mathbf{k}_m\right) \mathbf{v}_m^\top}{\cancel{\phi(\mathbf{q}_n)} \sum_{m \in M_c} \phi\left(\mathbf{k}_m\right)} = \frac{\sum_{m \in \mathcal{M}_c} \phi(\mathbf{k}_m) \mathbf{v}_m^\top}{\sum_{m \in \mathcal{M}_c} \phi\left(\mathbf{k}_m\right)}$$

- Learnable weights stabilize training

$$\boldsymbol{\phi}(\mathbf{x}) = \exp\left(\mathbf{x}^\top \boldsymbol{\omega}_{\boldsymbol{\phi}} - \frac{1}{2}\|\mathbf{x}\|^2\right)$$

[1] Choromanski, Krzysztof et al. "Rethinking Attention with Performers." ICLR 2021

EVA

$$\sum_{c=1}^{C} \frac{\exp\left(\mathbf{q}_n^\top \underbrace{\sum_{m\in\mathcal{M}_c} \alpha_m \mathbf{k}_m}_{\widetilde{\mathbf{k}}_c}\right)}{\sum_{c'=1}^{C} \exp\left(\mathbf{q}_n^\top \underbrace{\sum_{m\in\mathcal{M}_c} \alpha_m \mathbf{k}_m}_{\widetilde{\mathbf{k}}_c}\right)} \underbrace{\frac{\sum_{m\in\mathcal{M}_c} \boldsymbol{\phi}(\mathbf{k}_m)\mathbf{v}_m^\top}{\sum_{m\in\mathcal{M}_c} \boldsymbol{\phi}\left(\mathbf{k}_m\right)}}_{\widetilde{\mathbf{v}}_c}$$

- Implementation
  - Natively chunking-based so friendly to FA-style tiling and sharding

Algorithm 2 EVA Forward

Input: Input $\{\mathbf{q}_n\}_{n=1}^M$, $\{\mathbf{k}_m\}_{m=1}^M$, $\{\mathbf{v}_m\}_{m=1}^M$, chunk size P, and number of singleton groups S;
Output: EVA Outputs $\{\mathbf{o}_n\}_{n=1}^M$;

▷ Pre-compute all chunk-level outputs
Chunk $\{1, 2, \ldots, M\}$ into $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_C\}$ with same size P;
for $c = 1, 2, \ldots, C$ **do** ▷ Run in parallel
  Calculate $\widetilde{\mathbf{k}}_c = \sum_{m\in\mathcal{M}_c} \alpha_m \mathbf{k}_m$;
  Calculate $\widetilde{\mathbf{v}}_c = \frac{\sum_{m\in\mathcal{M}_c} \boldsymbol{\phi}(\mathbf{k}_m)\mathbf{v}_m^\top}{\sum_{m\in\mathcal{M}_c} \boldsymbol{\phi}(\mathbf{k}_m)}$;
for $n = 1, 2, \ldots, M$ **do** ▷ Run in parallel
  ▷ Neighborhood S elements to n are singleton groups
  ▷ Remaining are grouped into size-P chunks
  Partition $\{1, 2, \ldots, n\}$ into C_n groups $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_C\}$;
  Get $\mathbf{o}_n = \sum_{c=1}^{C_n} \frac{\exp(\mathbf{q}_n^\top \widetilde{\mathbf{k}}_c)}{\sum_{c'=1}^{C_n} \exp(\mathbf{q}_n^\top \widetilde{\mathbf{k}}_c)} \widetilde{\mathbf{v}}_c$;
Return $\mathbf{o}$.

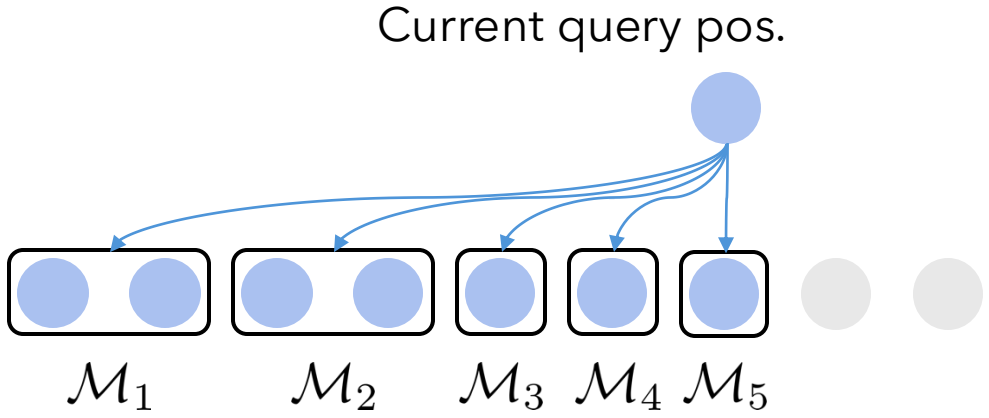
In EvaByte
Sequence length $M = 32768$
Singleton groups $S = 2048$
Fixed chunk size $P = 16$

EVA

In EvaByte
Sequence length $M = 32768$
Singleton groups $S = 2048$
Fixed chunk size $P = 16$

Compressive Decoding

- Behaves like standard attention
- Except that every S bytes
  - Pop most recent-S bytes from cache
  - Transform them into local memory slots
  - Append back to the cache



Algorithm 4 EVA Decoding

Input: Input $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$, EVA-cache, S', and S.
Output: Attention output $\mathbf{o}_t$.

▷ S is the number of total singleton groups
▷ $0 \le S' \le S$ is the number of current singleton groups in EVA-cache

if $S' = S$ **then**
  Evict most recent S elements $\left(\{\mathbf{k}_s\}_{s=1}^{S}, \{\mathbf{v}_s\}_{s=1}^{S}\right)$ from EVA-cache;
  Chunk $\{1, 2, \ldots, S\}$ into C groups $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_C\}$;
  for $c = 1, 2, \ldots, C$ **do**
    Calculate $\widetilde{\mathbf{k}}_c = \sum_{m \in \mathcal{M}_c} \alpha_m \mathbf{k}_m$;
    Calculate $\widetilde{\mathbf{v}}_c = \frac{\sum_{m \in \mathcal{M}_c} \boldsymbol{\phi}(\mathbf{k}_m)\mathbf{v}_m^\top}{\sum_{m \in \mathcal{M}_c} \boldsymbol{\phi}(\mathbf{k}_m)}$;
  Extend $\left(\{\widetilde{\mathbf{k}}_c\}_{c=1}^{C}, \{\widetilde{\mathbf{v}}_c\}_{c=1}^{C}\right)$ to EVA-cache;
  Reset $S' \leftarrow 0$;
else
  Append $(\mathbf{k}_t, \mathbf{v}_t)$ to EVA-cache;
  Increment $S' \leftarrow S' + 1$;

Read all C_t elements $\left(\{\widetilde{\mathbf{k}}_c\}_{c=1}^{C_t}, \{\widetilde{\mathbf{v}}_c\}_{c=1}^{C_t}\right)$ from EVA-cache;

Return $\mathbf{o}_t = \sum_{c=1}^{C_t} \frac{\exp\left(\mathbf{q}_t^\top \widetilde{\mathbf{k}}_c\right)}{\sum_{c'=1}^{C_t} \exp\left(\mathbf{q}_t^\top \widetilde{\mathbf{k}}_c\right)} \widetilde{\mathbf{v}}_c$.

EvaByte: Scalable Byte Modeling

- Byte modeling with streamlined arch.
  - No tokenization
  - Efficient attention – EVA
  - Multibyte prediction
- Scaling
  - 6.5B params + 32k context length + 1.5T bytes (~0.5T tokens)
- Strong performance
  - Faster decoding
  - Great data efficiency
  - No tokenization quirks
  - Native support to multimodal data types

Pretraining - Data

- Staged pretraining on SambaNova's hardware (SN-30 RDUs)
  - Phase 1 over 700B (LR cosine decay from 3e-4 to 2e-4):
    - 30% Fineweb-edu
    - 40% Dolma v1.7
    - 30% Stack v2
  - Phase 2 over 520B (LR cosine decay from 2e-4 to 1e-4):
    - 68% DCLM
    - 15% Dolma v1.7
    - 15% (Stack v2 + Opencoder)
    - 2% instruction data (FLAN & Open-math-instruct2)
  - Phase 3 over 100B + 200B (LR linearly decay from 1e-4 to 0):
    - Two independent annealing runs and model soup
    - 200B: 75% DCLM, 16% (Stack v2 + Opencoder), 9% instruction data
    - 100B: 50% DCLM, 25% (Stack v2 + Opencoder), 25% instruction data

Pretraining - Instability

- Embedding collapse
  - Temporary and weird typos
  - Self-resolved after a few training steps
- Loss spikes
  - We've made hyper-param. changes mid-flight
  - Most useful tips we found:
    - Lower adam_eps to 1e-12
    - Skipping batches
    - Periodically resetting optimizer states

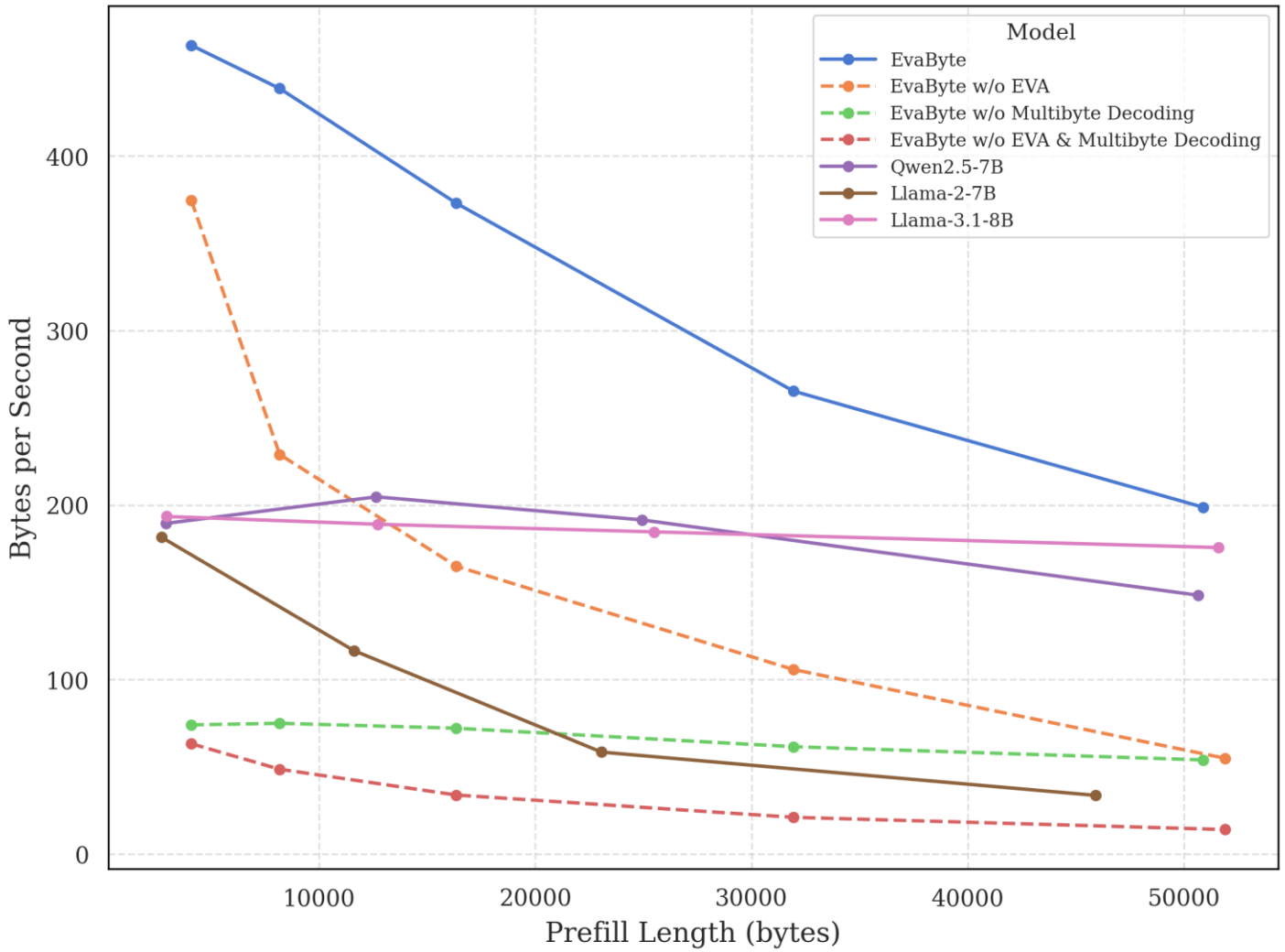
```
from typing import List, Tuple

def sum_product(numbers: List[int]) -> Tuple[int, int]:
    """ For a given list of integers, return a tuple co
all the integers in a list.
    Empty sum should be equal to 0 and empty product sh
    >>> sum_product([])
    (0, 1)
    >>> sum_product([1, 2, 3, 4])
    (10, 24)
    """
    sum = 0
    product = 1
    for number in numbirs:
        sum += numbir
        product *= numbir
    return (sum, product)
```

Results – Speed Up

- Both multibyte pred. and EVA speed up byte models
- 2x faster decoding than tokenizer-based LMs

(speed measured through native HF generate interface with FA2)



Results - Performance

Byte models
- More gradient descent steps with the same amount of data

0.5T tokens = 1.5T bytes

	Pythia	Amber	OLMo	Llama 2	OLMo-1.7	EvaByte	DCLM	MAP-Neo	OLMo-2	Llama 3	Gemma 2	Qwen 2.5
# Param.	6.9B	7B	7B	7B	7B	6.5B	7B	7B	7B	8B	9B	7B
# Tokens	0.3T	1.3T	2.5T	2.0T	2.5T	0.5T‡	2.5T	4.5T	4.0T	15.6T	8.0T	18.0T
Benchmark												
BoolQ	68.7	70.3	78.7	86.1	85.9	79.5	86.8	84.2	85.8	87.5	89.7	89.3
PIQA	74.9	78.0	78.5	77.5	80.3	74.1	80.1	77.4	80.7	81.6	85.0	87.7
SIQA	51.7	54.8	56.5	59.6	76.1	71.7	71.2	73.3	74.1	70.2	75.7	78.4
HellaSwag	66.1	74.4	78.1	78.9	80.1	68.7	82.3	72.9	83.8	81.8	86.5	89.9
WinoGrande	62.3	65.0	68.5	71.7	73.6	64.0	77.1	68.4	77.8	76.2	78.0	74.1
ARC-E	72.6	75.4	78.9	84.0	83.6	82.5	92.4	90.5	91.0	92.4	95.4	96.3
ARC-C	44.6	43.6	46.4	54.2	66.9	64.6	79.9	76.6	79.9	79.3	89.4	89.8
OpenbookQA	50.4	49.0	55.8	57.8	68.6	61.4	79.4	73.4	80.6	77.2	88.2	90.4
CommonSenseQA	62.1	67.6	70.8	74.2	85.8	80.1	77.0	80.7	75.1	73.9	78.4	86.2
MMLU	37.7	37.8	40.5	46.2	54.4	50.6	64.3	58.0	63.6	66.6	70.3	74.3
HumanEval-Plus	8.5	12.8	9.8	11.6	15.2	40.2	18.9	20.1	12.8	29.3	31.7	47.6
MBPP-Plus	13.2	21.2	21.2	22.8	25.7	51.1	6.1†	29.9	1.9†	51.6	53.2	63.8
GSM8K	3.4	5.4	5.8	14.6	28.4	41.1	46.6	53.7	67.5	56.0	70.7	85.4
MATH	3.8	2.8	1.6	2.5	4.9	19.5	14.7	20.7	17.6	20.5	37.7	48.9
Average	44.3	47.0	49.4	53.0	59.2	60.6	62.6	62.8	63.7	67.4	73.6	78.7

Results – Ablation

- EVA achieves same task perf. as standard attention
- Byte models catch up with tokenizer-based LMs
  - With 3x less data
- Or much better with the same amount of data

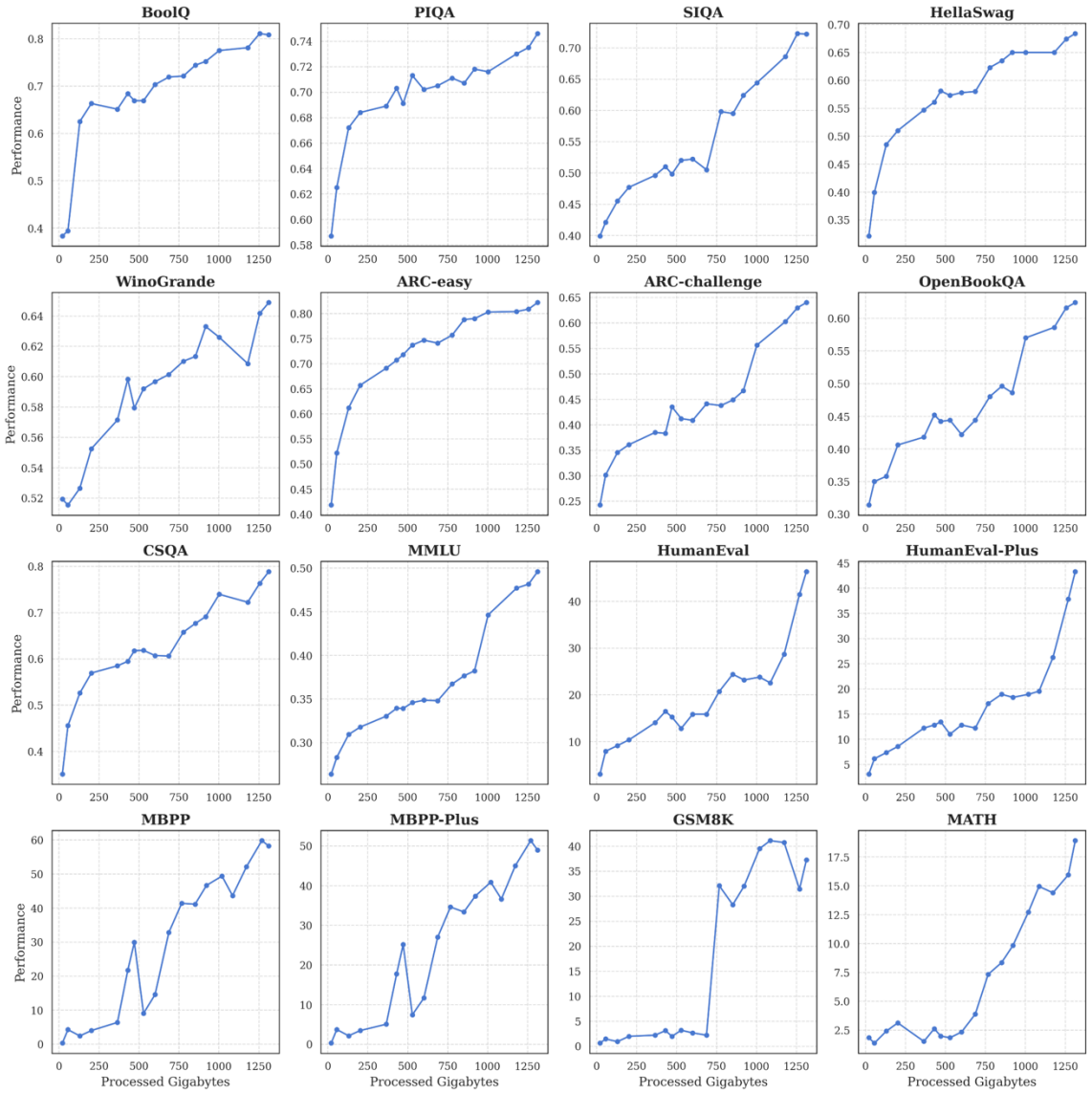
Tokenized models

Byte models

Task	Base LM 974M 100B Tokens	Base LM 974M 35B Tokens	EvaByte 874M w/o EVA 100B Bytes	EvaByte 874M 100B Bytes
HumanEval	14.0	7.3	11.0	11.6
HumanEval-Plus	11.0	6.1	9.8	9.1

Results – Intermediate Perf

- Downstream task perf. improves steadily throughout pretraining
  - No signs of plateauing yet



Results – Intermediate Perf

- Also outperforms open-sourced intermediate checkpoints trained on the same amount of data

Model	Tokens	BoolQ	PIQA	SIQA	HSwag	WinoG	ARC-e	ARC-c	OBQA	CSQA
OLMo-1.7-7B	2.5T	85.9	80.3	76.1	80.1	73.6	83.6	66.9	68.6	85.8
OLMo-2-7B	4.0T	85.8	80.7	74.1	83.8	77.8	91.0	79.9	80.6	75.1
OLMo-1.7-7B	0.4T	70.5	76.0	53.3	71.2	64.2	70.9	41.8	47.2	64.3
OLMo-2-7B	0.4T	76.7	78.1	59.0	76.2	68.7	80.1	52.1	56.0	70.7
EvaByte (Before annealing)	1.22T bytes	79.2	72.7	68.1	67.9	62.6	79.8	61.4	58.4	75.8

Model	Tokens	MMLU	HumanEval	HumanEval-Plus	MBPP	MBPP-Plus	GSM8K	MATH	Average*
OLMo-1.7-7B	2.5T	54.4	17.1	15.2	32.3	25.7	28.4	4.9	59.2
OLMo-2-7B	4.0T	63.6	14.6	12.8	2.1†	1.9†	61.3	17.6	63.3
OLMo-1.7-7B	0.4T	36.2	10.4	9.1	3.7	3.2	3.4	2.1	43.8
OLMo-2-7B	0.4T	43.8	5.5	4.9	2.1	2.1	7.3	2.6	48.4
EvaByte (Before annealing)	1.22T bytes	48.2	37.8	34.1	52.4	45.5	38.4	13.7	57.6

Results - SFT

- Byte models also scale with SFT
  - Our final mix uses TULU v3 and filtered Opencoder

Model	BBH	GSM8k	IFEval	MATH	MMLU	HumanEval*	TruthQA
Gemma-2-9B-it	20.0	79.7	69.9	29.8	69.1	71.7	61.4
Ministral-8B-Instruct	56.2	80.0	56.4	40.0	68.5	91.0	55.5
Qwen-2.5-7B-Instruct	25.3	83.8	74.7	69.9	76.6	93.1	63.1
Llama-3.1-8B-Instruct	69.7	83.4	80.6	42.5	71.3	86.3	55.1
Tülu 3 8B	66.0	87.6	82.4	43.7	68.2	83.9	55.0
OLMo-7B-Instruct	35.3	14.3	32.2	2.1	46.3	28.7†	44.5
OLMo-v1.7-7B-Instruct	34.4	23.2	39.2	5.2	48.9	49.7†	55.2
OLMoE-1B-7B-0924-Instruct	37.2	47.2	46.2	8.4	51.6	54.8	49.1
MAP-Neo-7B-Instruct	26.4	69.4	35.9	31.5	56.5	72.1†	51.6
OLMo-2-7B-SFT	50.7	71.2	68.0	25.1	62.0	67.0†	47.8
OLMo-2-7B-1124-Instruct	48.5	85.2	75.6	31.3	63.9	67.6†	56.3
EvaByte-SFT	34.6	52.9	60.2	29.8	49.5	73.7	46.3

Results – Tokenization Quirk Fix

- Tokenization issues: prompt boundary problem

prompt | correct completion | incorrect completion

Qwen2.5-7B: different prompt boundaries lead to diverging and unexpected outputs.

```
▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    ">>> longest([\'a\', \'bb\', \'ccc\', \'dddd\'])\n
\'dddd\'\n    """\n    i...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """"\n    if not strings:\n        return None\n
longest_string =...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n  if not strings:\n    return None\n  longest =
strings[0]\n  for st...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n  # if not strings:\n    #    return None\n    # longest
= strings[...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n
longest_string = string...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if len(strings) == 0 None:\n        return None\n
else:\n      ...
```

EvaByte: outputs from different prompt boundaries converge.

```
▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]\n...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]\n ...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]\n  ...

▶ def longest(strings: List[str]) -> Optional[str]:\n    """ Out of list of strings, return the longest one.
...>>> longest(['a', 'bb', 'ccc'])\n    'ccc'\n    """\n    if not strings:\n        return None\n    longest =
strings[0]\n   ...
```

Results - Robustness

- But byte models still struggle with tasks such as character counting, string processing, etc.
- Likely not a problem of tokenization but more about reasoning or representation issues
  - Models need deeper semantic understanding to retrieve the correct information from context

User:
What is the third letter in the string “hkunlp"?

EvaByte-SFT:
The third letter in "hkunlp" is “n". ✗

Results - Multimodality

- Byte models are flexibly extendable to various data types
- Just as text, images can take any byte representation with some encoding
- Due to 32KB context length limit, we use JPEG
  - 256 x 256 image ~ 10K bytes
  - This choice is suboptimal: JPEG is lossy and optimized for human perception not for machines

Results - Multimodality

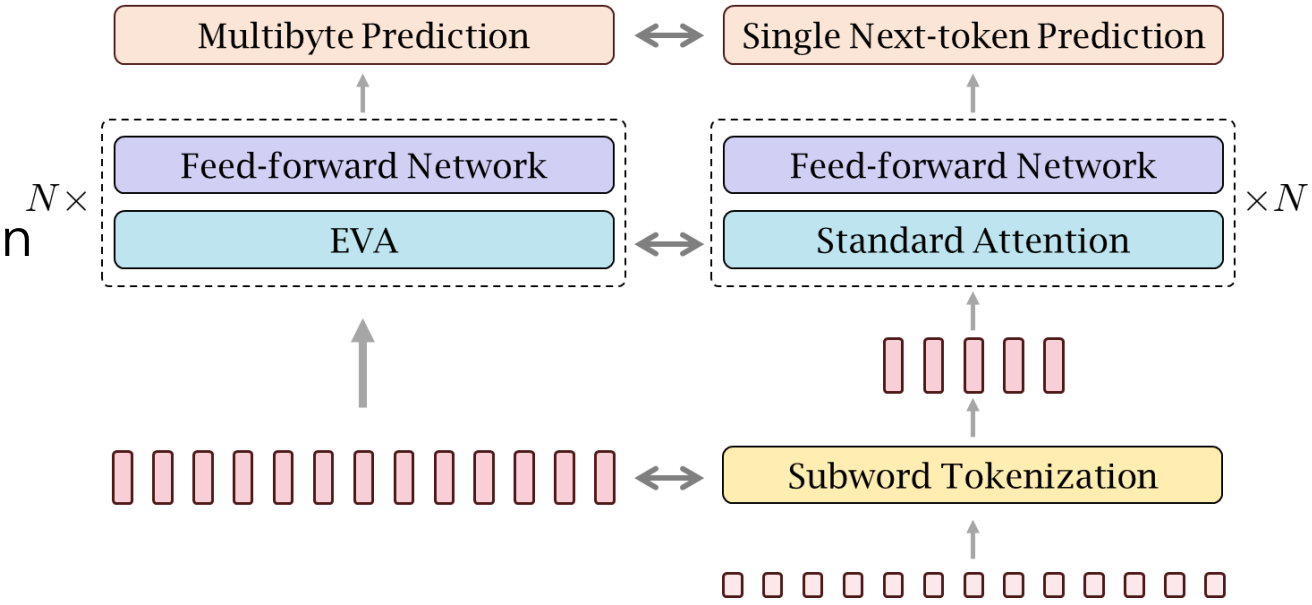
- Directly feed interleaved image + text bytes
  - without any arch. changes
- After finetuning 20k steps on image-caption pairs
  - Vaguely reconstructs images
  - Captions with reasonable detail

Image	Text
	**Detailed caption:** The image shows a close-up view of a laptop keyboard. The keyboard has a black color scheme with white lettering on the keys. The keys are arranged in a standard QWERTY layout, which is commonly used for English language keyboards. The keys are labeled with the standard Western letters and symbols, including the alphabet, numbers, and punctuation marks. The laptop appears to be a modern model, possibly a MacBook, given the design and the visible trackpad to the right of the keyboard. The trackpad is a flat, rectangular surface with a grid pattern, typical of MacBook trackpads. The laptop is open, and the keyboard is the main focus of the image. The background is blurred, but it seems to be an indoor setting with a wooden surface, possibly a table or desk. The lighting in the image is soft, suggesting an indoor environment with ambient lighting.
	**Detailed caption:** The image shows a black dog swimming in a body of water. The dog appears to be a medium to large breed, with a glossy black coat. It is captured mid-swim, with its head above the water and its body partially submerged. The water around the dog is rippling, indicating movement, and the surface of the water reflects the light, suggesting it is a sunny day. There are no visible texts or distinguishing marks that provide additional context about the location or the dog's identity. The style of the image is a candid, real-life photograph, capturing a moment of the dog's activity.
	**Q:** How many throw pillows are on the bed? **A:** three

Limitations & Future Work

Our tweaks to the Transformer so far:

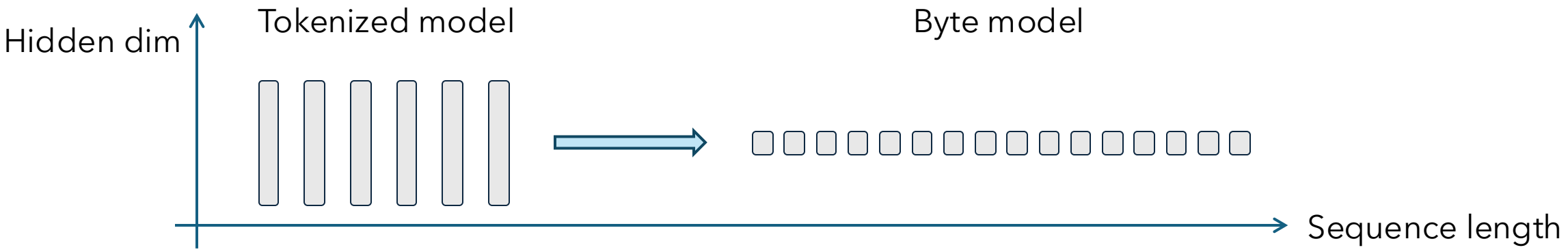
- Input
  - tokenized -> byte-level
- Output
  - Next-token -> multi-byte prediction
- Attention
  - Standard attention -> EVA
- FFN
  - Remains the same
  - Remains the bottleneck esp. with efficient attention modules
  - Improving FFNs' efficiency?



[1] Geva, Mor, et al. "Transformer feed-forward layers are key-value memories." EMNLP 2021.

Limitations & Future Work

- Data representation: the model can take any byte stream
  - Many valid choices to represent the same piece of data
  - Modeling over either raw bytes or compressed encodings
- Model architecture
  - Toward more distributed representation learning
- Bytes look like a suitable testbed for efficient sequence model research



Thank you!