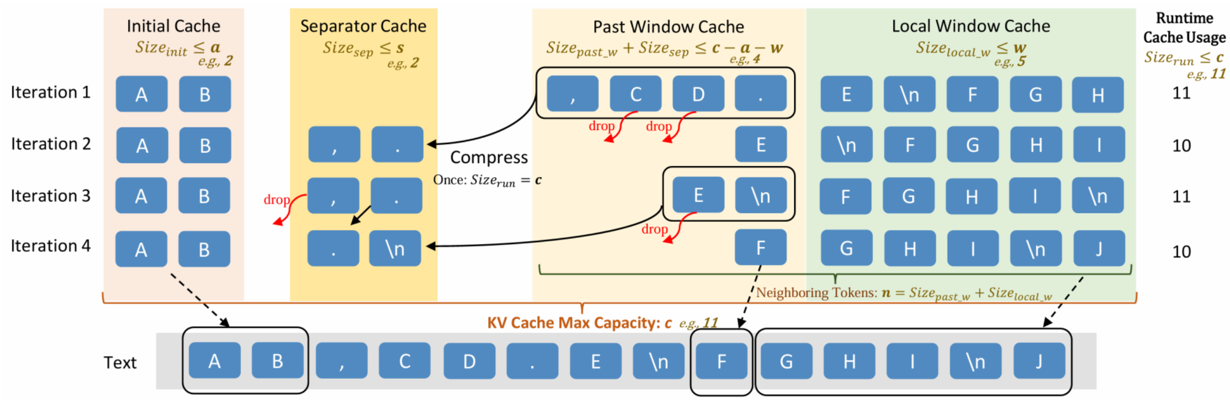
SepLLM: Accelerate Large Language Models by Compressing One Segment into One Separator

An Easy-to-Use Native Sparse Attention Baseline Method

sepllm.github.io

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, Chao Huang



Noah's Ark Lab, Huawei Technologies

The University of Hong Kong

Center of Excellence in Generative AI, KAUST

Max Planck Institute for Intelligent Systems, Tübingen

Background

- The attention mechanism has quadratic complexity, and the KV cache grows linearly with the text length.
- The size of the KV cache being too large can affect inference speed and consume a significant amount of GPU memory, especially for long-text tasks.
- Most existing training-free methods are query-dependent: filtering the most relevant KV based on the current query.
- The existing sparse attention baseline methods are overly sparse. (*e.g.*, *StreamingLLM[arXiv:2309.17453]*)

Note: **SepLLM is suitable to serve as the <u>fundamental baseline model for sparse attention mechanisms</u> in LLMs.**

Observation

- An interesting pattern: certain seemingly meaningless **separator tokens** (i.e., punctuations) contribute disproportionately to attention scores compared to semantically meaningful tokens.
- >>> **Information of the segments** between these separator tokens can be effectively condensed into the **separator tokens themselves**

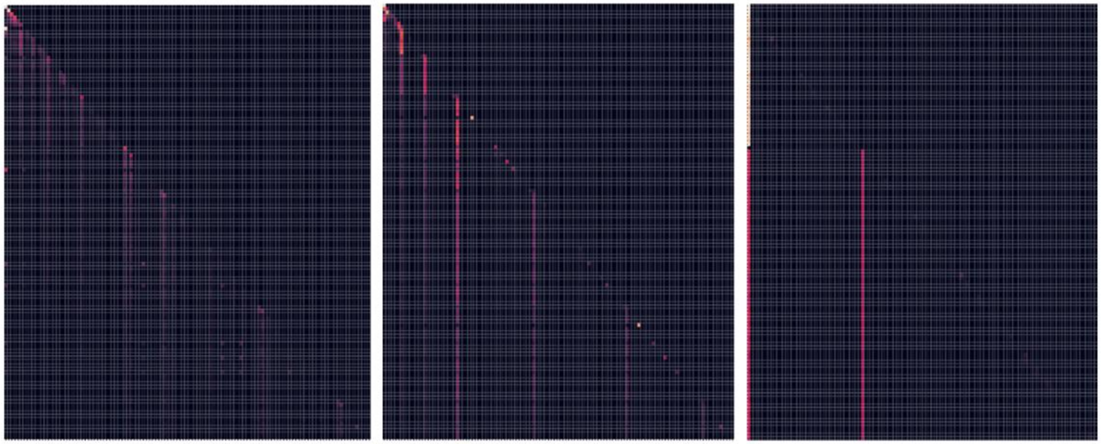


Figure 2. The visualization for attention scores of different layers given the input "Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. ...". Note that the separator tokens like "," and "." contribute massive attentions.

Fundamental Design

- During Training & Pre-filling: only attend to *Initial Tokens*, *Separator Tokens*, *Local Tokens*
- During Inference: decode based on the KV pairs of *Initial Tokens*, *Separator Tokens*, *Local Tokens*

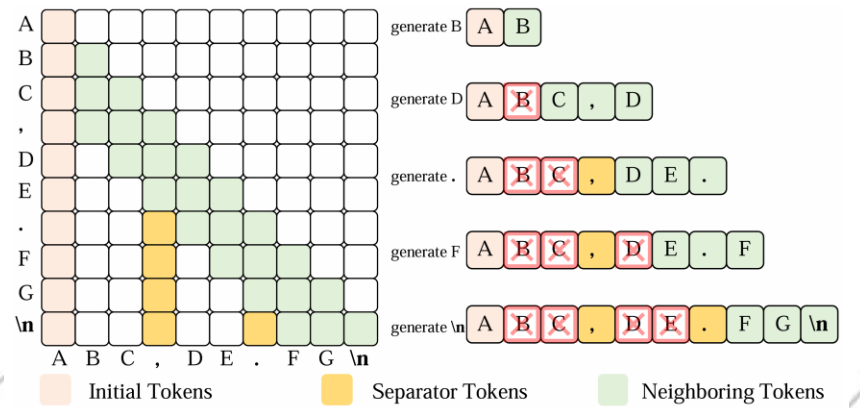


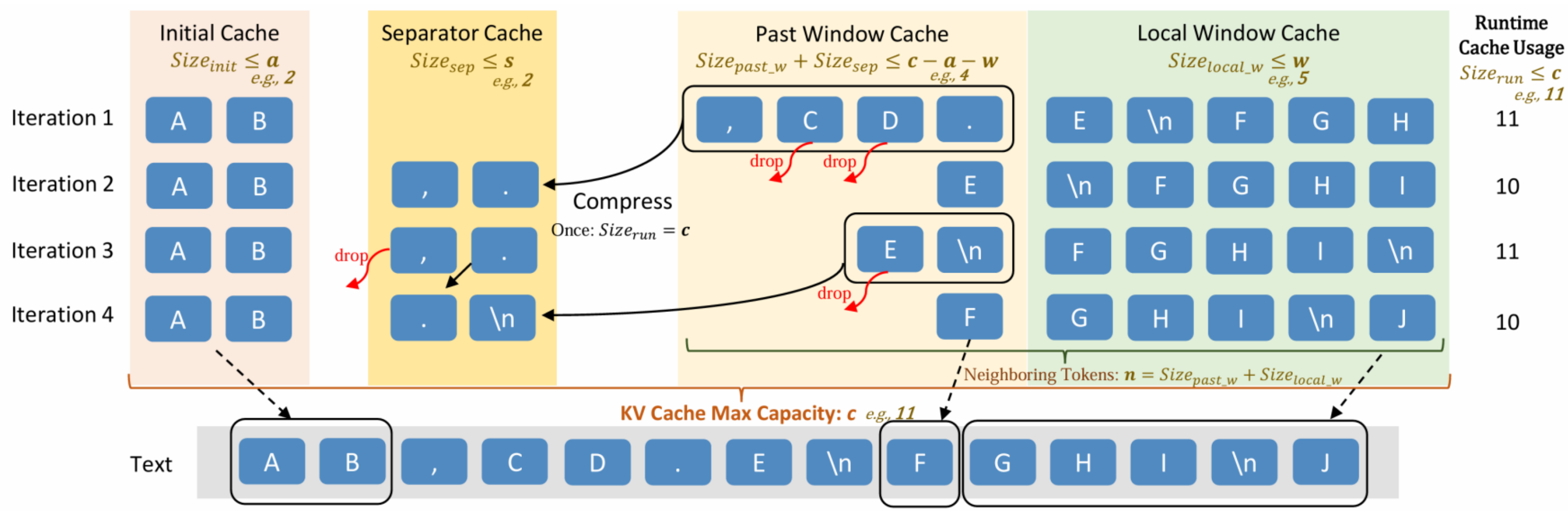
Figure 3. The overall paradigm of SepLLM. The left side illustrates the attention mask in the training or pre-filling stage given the input "ABC,DE.FG$\backslash n$". The right side illustrates the KV cache management in the generation stage.

SepLLM closely aligns with the semantic distribution of natural language because the separator itself provides a division and summary of the current segment. The segments separated out are inherently semantically coherent, forming self-contained semantic units.

Note: During the pretraining phase, **SepLLM** ***intentionally*** compresses segment information into the separator used to divide the segment.

Tailored Streaming Design

- To facilitate scenarios of streaming inference with infinitely long inputs (where separators' KVs may accumulate indefinitely) and to simplify KV cache management, we propose the following design.



Experiments

- **Entire Lifecycle of LLM**: optimization for *training-from-scratch*, *post-training*, and *training-free*.
- **Validation on Large-Scale Data**: PILE dataset (over 300B tokens)
- **Adaptation to Different Inference Lengths**: From <2k, to 20K, 4M.
- **Adaptation to Different Backbone Architectures and Sizes**: Pythia-160M, Pythia-1.4B, Pythia-6.9B, Pythia-12B, Llama3-8B, GPTNeoX-20B, and Falcon-40B, *etc*.
- **Extensive Downstream Task Benchmarking**: our evaluation covers 15+ challenging benchmarks across 5 capability dimensions:

Capability Dimensions	Tasks
Knowledge Reasoning	MMLU, ARC-Easy, ARC-Challenge
Mathematical Reasoning	GSM8K_CoT, LogiQA, **MATH(challenging)**
Linguistic Understanding	LAMBADA(OpenAI), PIQA, WinoGrande, WSC, Pile
Scientific Comprehensiong	SciQA, MMLU
Long-Context Processing	Needle In A Haystack, PG19, Wikitext

Experimental Results

- Training-Free Results

	GSM8K-CoT			MMLU					
	flexible	strict	*r*.KV (%)	humanities	stem	social	other	Overall	*r*.KV (%)
Vanilla	77.79	77.26	100.00	60.49	56.61	76.50	72.19	65.72	100.00
StrmLLM (***n***=380)	70.89	71.42	47.54	57.73	54.46	74.39	70.13	63.39	52.50
StrmLLM (***n***=256)	69.67	68.61	26.00	62.10	54.49	73.06	69.78	62.10	37.73
SepLLM (***n***=256)	77.18	77.18	47.36	57.66	56.49	76.21	72.19	64.68	44.61

Table 1. Evaluation results and average *runtime* KV cache usage for training-free experiments on GSM8K-CoT 8-shots and MMLU 5-shots. For SepLLM and StreamingLLM, three initial tokens' KV are kept for this experiment. *r*.KV (%) here represents the ratio of KV usage at *runtime* for the respective method compared to Vanilla. See more results in Appendices I and Table 17.

- Downstream Results of Trained SepLLM Models:

Method	ARC-c	ARC-e	LBD-ppl	LBD-acc	LogiQA	PIQA	SciQ	Atten. (%)	*r*.KV (%)
Vanilla	20.14	46.80	34.83	33.28	23.81	62.84	81.50	100.00	100.00
StrmLLM(***n***=64)	20.65	47.39	44.03	26.74	21.97	63.82	75.80	16.58	15.28
SepLLM(***n***=64)	19.62	46.46	40.08	28.97	26.42	63.82	80.10	25.83	25.40
SepLLM(***n***=128)	19.97	47.35	30.16	33.18	22.73	64.64	82.60	35.64	32.27
SepLLM(***n***=64,H)	20.73	48.44	36.54	30.45	25.35	64.36	80.60	32.01	31.58
SepLLM(***n***=64,H/T)	21.42	47.26	33.41	32.80	22.73	63.98	81.20	38.18	37.75

Table 2. The performance of downstream tasks and the average *runtime* KV cache usage in the training-from-scratch setting.

Note: You are recommended to train from scratch to achieve the optimal performance of SepLLM

Training Process

- Post-Training

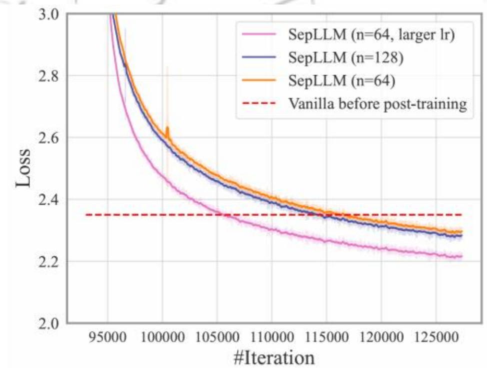
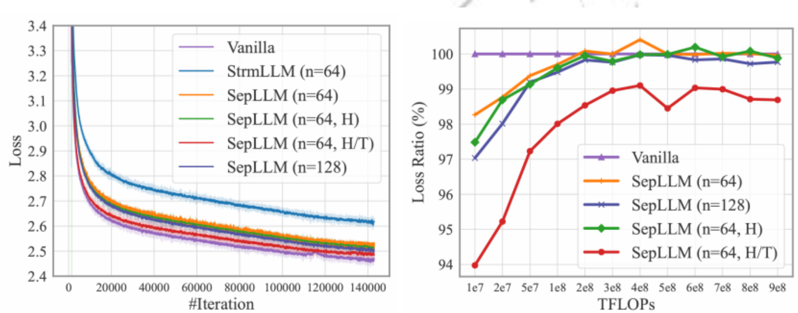


Figure 6. Training loss curves for the post-training setting.

- Training-from-Scratch



(a) Loss *w.r.t* steps (b) Loss Ratio *w.r.t* FLOPs

Figure 5. Training loss curves for training from scratch. 5(b) shows the ratios of the loss values of different methods to that of Vanilla with respect to FLOPs.

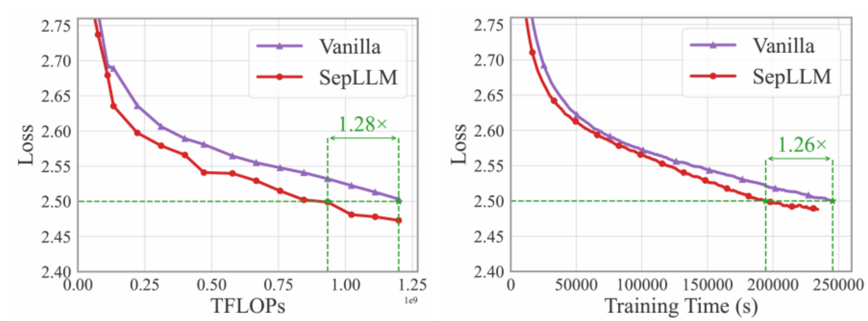


Figure 1. The loss comparison between vanilla Transformer and the proposed SepLLM. SepLLM achieves lower loss *w.r.t* different computation costs and different training time consistently.

Long-Streaming Test

- Lower perplexity, less KV cache, less inference time based on different models, sizes, *etc*.

PG19	1M	1.5M	2M	2.5M	3M	3.5M	4M
StrmLLM	39.5	38.2	38.3	37.6	36.4	35.8	36.1
SepLLM (*s*=32)	37.7	36.6	36.6	36.0	34.9	34.2	34.5
SepLLM (*s*=64)	37.1	36.0	36.1	35.4	34.3	33.7	33.9

Table 4. The perplexity comparison on the PG19 test set (Rae et al., 2020). For fair evaluation, we keep the entire KV cache capacity ***c*** as 324 and Initial Cache capacity ***a*** as 4 for both StreamingLLM and SepLLM. ***w***=224, ***s***=32/64 for SepLLM.

Length	Methods	***c***	*r*.KV	ppl	time (s)
20K	Vanilla	20K	10K	302.6	523.8
	StrmLLM	800	800	31.5	341.2
	SepLLM	800	562	28.3	325.8
64K	Vanilla	64K	32K	1090.8	3380.6
	StrmLLM	800	800	37.9	1096.0
	SepLLM	800	562	33.4	1049.7

Table 5. The average perplexity and running time comparison on the PG19 test set (Rae et al., 2020). *r*.KV means the average *runtime* KV cache usage in the generation process.

Length	Methods	***c***	*r*.KV	ppl	time (s)
20K	StrmLLM	1024	1024	8.98	1512.88
	StrmLLM	800	800	9.02	1430.59
	SepLLM	1024	906	8.92	1440.89
	SepLLM	800	690	9.00	1368.07
64K	StrmLLM	1024	1024	11.01	4844.79
	StrmLLM	800	800	11.09	4623.90
	SepLLM	1024	906	10.96	4619.63
	SepLLM	800	690	11.07	4414.72

Table 13. The comparison of SepLLM adapted to Falcon-40B (Almazrouei et al., 2023).

Backbone	Arch.	***c***	*r*.KV	ppl	time(s)
Pythia-6.9B	Vanilla	64K	32K	1037.6	4160.7
	StrmLLM	800	800	15.9	1522.6
	SepLLM	800	562	15.8	1456.0
Llama-3-8B	Vanilla	64K	32K	1090.8	3380.6
	StrmLLM	800	800	37.9	1096.0
	SepLLM	800	562	33.4	1049.7

Backbone	***a***	***s***	***w***	***c***	*r*.KV	ppl	time(s)
Pythia-6.9B	4	64	256	800	562	13.0	445.0
	4	64	800	1024	946	12.7	450.4
	4	64	928	1280	1138	12.7	454.4
Pythia-12B	4	64	256	800	562	12.1	577.0

Table 12. The comparison of SepLLM adapted to Pythia (Biderman et al., 2023) with different scales.

Ablation Study

- On local size (*w*) and whole cache size (*c*)

Method	*w*	*c*	*r*.KV	5K	10K	15K	20K
StrmLLM	320	324	324	13.18	11.51	8.85	8.91
	512	516	516	12.87	11.37	8.74	8.78
	796	800	800	11.96	11.01	8.67	8.72
SepLLM	224	324	308	13.01	11.17	8.67	8.72
	320	516	452	12.91	11.26	8.67	8.72
	512	800	690	12.09	11.03	8.56	8.62

Table 7. Average downstream performance (ppl) over different input lengths and average *runtime* KV usage with different ***c***,***w*** on WikiText, in which ***a***=4 for both methods and ***s***=64 for SepLLM.

- On initial tokens and positional encoding shifting (PE shifting).

Method	initial	shift	5K	10K	15K	20K	*r*.KV
StrmLLM	✓	✓	13.2	11.5	8.9	8.9	324
StrmLLM	✗	✓	14.6	13.2	10.8	10.9	324
StrmLLM	✓	✗	425.5	513.1	509.5	506.8	324
StrmLLM	✗	✗	409.4	540.5	527.5	558.2	324
SepLLM	✓	✓	13.1	11.3	8.7	8.8	292
SepLLM	✗	✓	14.9	14.3	12.4	12.5	290
SepLLM	✓	✗	192.7	214.6	175.0	174.4	292
SepLLM	✗	✗	226.4	264.7	227.5	228.8	290

Table 8. The perplexity and average *runtime* KV cache usage of SepLLM and StreamingLLM tested on WikiText (Merity et al., 2017). ***c***=324, ***a***=0/4 for both methods. ***s***=32,***w***=224 for SepLLM

Note: In practice, **no need** to do PE shifting if the actual length does not exceed the pretrained max PE length.

Ablation Study

- On Choice of Separators

	SepLLM (n=256)	SepLLM (***n***=256)	SepLLM (***n***=256)	StrmLLM (***n***=256)	StrmLLM (***n***=380)
Separators	“.” “,” “?” “!” “;” “:” “ ” “\t” “\n”	“,” “.” “?” “;”	“.” “?”	None	None
r.KV (%)	47.36	37.92	36.44	31.92	47.54
flexible-extract	77.18	76.68	70.66	68.84	71.42
strict-match	77.18	76.85	70.36	67.63	70.89

Table 16. Evaluation results and average *runtime* KV cache usage for training-free experiments on GSM8K-CoT with 8-shots, based on different choices of separators.

- Fixed-Interval Variant (*FixLLM*)

	GSM8K-CoT			MMLU					
	flexible	strict	*r*.KV (%)	humanities	stem	social	other	overall	*r*.KV (%)
Vanilla	77.79	77.26	100.00	60.49	56.61	76.50	72.19	65.72	100.00
FixLLM (Δ=5, ***n***=256)	70.43	70.43	45.64	55.52	54.80	72.99	69.75	62.33	50.20
FixLLM (Δ=4, ***n***=256)	72.71	72.33	49.08	55.92	54.39	74.36	70.81	62.91	53.32
SepLLM (***n***=256)	77.18	77.18	47.36	57.66	56.49	76.21	72.19	64.68	44.61

Table 17. Evaluation results and average *runtime* KV cache usage for training-free experiments on GSM8K-CoT 8-shots and MMLU 5-shots. For SepLLM and FixLLM, three initial tokens' KV are kept. Δ denotes the interval size for FixLLM and ***n*** is the number of retained neighboring tokens' KV. *r*.KV (%) represents the ratio of KV usage at *runtime* for the respective method compared to Vanilla.

Note: See many other experimental results (e.g., *Needle in a Haystack*) in the paper

Source Code and Usage

- You can find our code at *https://github.com/HKUDS/SepLLM*
- Or: *sepllm.github.io*

- You can find all the code related to *training-free*, *streaming*, and *training-from-scratch* experiments.

If you find our code useful, please consider giving us a star 🌟
Your support is greatly appreciated 😊

Demo Usage

- To run this `SepCache` demo, you must install our `transformers` package from our repository: *https://github.com/HKUDS/SepLLM*

```
from transformers import AutoTokenizer, AutoModelForCausalLM, SepCache
import torch
from huggingface_hub import login
login(“xxxXXXxxx")


def to_cuda(a_dict: dict) -> dict:
    new_dict = {}
    for k,v in a_dict.items():
        if isinstance(v, torch.Tensor):
            new_dict[k] = v.cuda()
        else:
            new_dict[k] = v
    return new_dict

model = AutoModelForCausalLM.from_pretrained("meta-llama/Meta-Llama-3-8B-Instruct", attn_implementation="flash_attention_2", device_map="cuda:0")
model.bfloat16().cuda()
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Meta-Llama-3-8B-Instruct")
inputs = tokenizer(text=["My name is Llama 3"], return_tensors="pt")
inputs = to_cuda(inputs)

past_key_values = SepCache(init_cache_size=4,sep_cache_size=128,local_size=256,cache_size=512, layer_num=32, USE_MAX_SEP_CACHE=True, model_type='llama')
outputs = model(**inputs, past_key_values=past_key_values, use_cache=True)
outputs.past_key_values # access cache filled with key/values from generation
```

- When using the `update` function of `SepCache` to update the keys/values and the past token ids (necessary in `SepCache`), the current `input_ids` must also be provided.

```
key_states, value_states, query_states = past_key_values.update(
    key_states = key_states,
    value_states = value_states,
    input_ids = input_ids,
    layer_idx = layer_idx,
    PREFILLING_FLAG = q_len > 1, ## `q_len` is the sequence length of the current `query_states`
    cache_kwargs = None )
```

End of Presentation

THANK YOU

Comments

Questions

Discussion

```
@inproceedings{chen2025sepllm,
  title={{SepLLM: Accelerate Large Language Models by Compressing One Segment into One Separator}},
  author={Chen, Guoxuan and Shi, Han and Li, Jiawei and Gao, Yihang and Ren, Xiaozhe and Chen, Yimeng and Jiang, Xin and Li, Zhenguo and Liu, Weiyang and Huang, Chao},
  booktitle={International Conference on Machine Learning},
  year={2025}
}
```