

A Inference Efficiency Analysis

To quantitatively assess the computational overhead introduced by our proposed method PaceLLM, we conduct a series of rigorous inference time measurements on the Qwen2-7B model using the Qasper task from LongBench—a representative long-context question answering benchmark. Our evaluation focuses on both absolute inference time and relative time increase compared to baseline methods. The results are summarized in Table 6.

Table 6: Inference Time Comparison on Qwen2-7B (Qasper Task)

Method	Sdpa Attention	Flash Attention
Vanilla	7m31s	7m03s
Activation Beacon	4m42s	4m19s
Ours	6m32s	6m09s

Controlled Time Overhead. Compared to the most efficient baseline (Activation Beacon), our method introduces a moderate and controlled increase in inference latency. Specifically, the relative time overhead is approximately $\times 1.37$ with SDPA and $\times 1.32$ with FlashAttention. However, compared to the Vanilla baseline without any memory mechanism, our method achieves a significant speedup—about 13.2% faster under SDPA and 13.4% faster under FlashAttention. This highlights that our approach strikes a favorable balance between computational complexity and memory-enhanced modeling capability.

Compatibility with Attention Optimizations. All methods benefit from attention-level optimization. Transitioning from SDPA to FlashAttention yields a consistent 6–7% speedup across all setups. Importantly, our method is fully compatible with FlashAttention, demonstrating its practical applicability to real-world, performance-critical environments.

Breakdown of Overhead Sources. The primary computational overhead in our method stems from the activation memory mechanism, including dynamic activation storage, similarity-based lookup, and selective activation reconstruction. These components are central to the model’s ability to capture and reuse long-range dependencies. Nonetheless, they are designed to be lightweight, ensuring that the overall throughput remains practical.

Efficiency–Performance Trade-off. The additional inference time is well justified by the performance gains observed in multiple long-context tasks. Compared to the Vanilla baseline, our method reduces latency while improving comprehension. Compared to the Activation Beacon, we achieve stronger results with acceptable overhead. For latency-sensitive applications, the design of our system offers a tunable trade-off between inference efficiency and accuracy.

Summary. PaceLLM maintains operational feasibility with predictable computational cost. It integrates well with widely adopted acceleration techniques such as FlashAttention and provides a favorable performance–efficiency trade-off, making it suitable for both research and real-world deployment scenarios.

B Detailed Performance on LongBench

Table 7 reports the performance of PaceLLM on a variety of long-context understanding tasks from LongBench in a training-free setting. We evaluate two major foundation models—Qwen-2 and Llama-2—and progressively apply our brain-inspired mechanisms: cortical expert neuron clustering (CE) and persistent activity memory (PA).

Component-wise Improvements. Individually, both CE and PA contribute positively across most tasks. For Qwen-2, CE enhances performance particularly in *Single-Document QA* (e.g., NrtvQA improves from 25.38 to 25.87) and *Code* tasks (e.g., RB-P rises from 46.47 to 46.71). PA, on the other hand, is especially effective in *Few-shot Learning* (e.g., TREC from 76.00 to 78.00) and

Table 7: Performance comparison between PaceLLM and baseline models on LongBench tasks in **training-free** manner. CE denotes cortical expert neuron clustering and PA means persistent activity memory mechanism.

	Method	Single-Document QA				Multi-Document QA				Summarization			Few-shot Learning			Code				
		NrtvQA	Qasper	MF-en	Avg.	HotpotQA	2WikiMQA	Musique	Avg.	GovReport	QMSum	MultiNews	Avg.	TREC	TriviaQA	SAMSum	Avg.	Lcc	RB-P	Avg.
Qwen-2	Vanilla	25.38	42.75	45.16	37.76	55.29	54.91	36.88	49.03	36.68	23.52	26.60	28.93	76.00	90.16	44.91	70.36	53.63	46.47	50.05
	Vanilla+CE	25.87	42.30	44.86	37.68	54.15	55.41	36.83	48.80	36.30	23.55	26.39	28.85	76.50	89.91	45.42	70.61	54.00	46.71	50.36
	Vanilla+PA	25.24	43.86	45.17	38.09	55.62	55.61	36.84	49.36	36.06	23.74	26.77	28.86	78.00	89.41	45.35	70.92	52.90	46.29	49.60
	Vanilla+PA+CE	26.15	43.88	45.45	38.49	56.53	56.31	37.99	50.28	36.54	23.59	26.92	29.02	78.00	89.41	45.46	70.96	53.76	46.13	49.95
Llama-2	Vanilla	16.65	19.77	35.34	23.92	34.27	26.90	9.08	23.42	26.54	20.85	25.90	24.43	64.50	83.34	41.21	63.02	58.59	52.38	55.48
	Vanilla+CE	16.90	20.30	36.26	24.49	35.11	27.51	8.58	23.73	26.34	21.11	25.69	24.38	64.00	83.34	41.24	62.86	58.07	52.27	55.17
	Vanilla+PA	17.76	20.82	35.36	24.65	33.33	27.31	8.81	23.15	25.99	20.96	25.60	24.18	65.00	83.42	41.28	63.23	58.23	51.73	54.98
	Vanilla+PA+CE	18.34	21.26	36.44	25.35	34.37	27.57	9.32	23.75	26.51	21.08	26.25	24.61	66.00	83.59	41.66	63.58	58.33	52.22	55.28

Multi-Document QA (e.g., HotpotQA from 55.29 to 55.62), aligning with its role in preserving longer contextual dependencies.

For Llama-2, the gains are also evident. CE improves complex QA tasks such as MF-en (from 35.34 to 36.26) and long-context comprehension tasks like 2WikiMQA. PA further boosts performance in NrtvQA and TREC. These results demonstrate that each mechanism targets complementary cognitive functions and boosts model reasoning in different ways.

Synergistic Combination. When CE and PA are combined, they consistently lead to the best overall performance across all categories and both models. Notably:

- For Qwen-2, *Multi-Document QA* tasks show the most significant gains: HotpotQA improves from 55.29 to 56.53, and Musique from 36.88 to 37.99. These tasks demand multi-hop reasoning and long-span memory, where our dual mechanisms work jointly to capture hierarchical and persistent context.
- In summarization tasks such as QMSum and MultiNews, CE+PA achieves or closely approaches the best results (e.g., MultiNews from 26.60 to 26.92).
- *Few-shot Learning* tasks also benefit, where CE+PA maintains the highest scores in TREC and SAMSum.

Cross-Model Robustness. Our approach generalizes well across architectures. Although Llama-2 starts from a lower baseline than Qwen-2, it benefits significantly from our enhancements:

- The CE+PA combination raises performance in NrtvQA by +1.69, Qasper by +1.49, and MF-en by +1.10 over vanilla Llama-2.
- *Multi-Document QA* and *Summarization* also show consistent gains (e.g., 2WikiMQA from 26.90 to 27.57, MultiNews from 25.90 to 26.25).
- Few-shot tasks exhibit either improved performance, indicating the method’s stability.

Summary. Overall, the experimental results underscore the effectiveness of our brain-inspired design. The CE mechanism enhances specialized, local processing by routing to expert neuron clusters, while PA extends the temporal memory span. Their integration leads to robust performance improvements across 15+ diverse tasks without any parameter update, setting a new standard for training-free long-context understanding. Notably, these results are achieved with minimal computational overhead (as discussed in Section A), ensuring practical deployment feasibility.

C Detailed Methodology of PaceLLM

C.1 Persistent Activity (PA)-Activation Working Memory Bank Operations

Algorithm 1 describes the working memory mechanism of PaceLLM, which dynamically enhances current FFN activations using a memory bank. It consists of three key phases: retrieval, enhancement, and memory update.

- **Input:** Activation tensor $\mathbf{X} \in \mathbb{R}^{B \times L \times d_{\text{ff}}}$ (where B is batch size, L is sequence length, and d_{ff} is FFN dimension), and a memory bank $\{\mathbf{K}, \mathbf{V}, \mathbf{u}\}$ storing previous activation keys, values, and usage counters.

Algorithm 1 Persistent Activity (PA)-Activation Working Memory Bank Operations

Require: Current activation $\mathbf{X} \in \mathbb{R}^{B \times L \times d_{\text{ff}}}$, memory bank $\{\mathbf{K}, \mathbf{V}, \mathbf{u}\}$

Ensure: Enhanced activation \mathbf{O} , Updated memory bank

```
1:  $\mathbf{X}_{\text{flat}} \leftarrow \text{Flatten}(\mathbf{X})$   $\{\mathbf{X}_{\text{flat}} \in \mathbb{R}^{(B \times L) \times d_{\text{ff}}}\}$ 
2: Initialize  $\mathbf{O}_{\text{flat}} \leftarrow \mathbf{0}$ 
3: for chunk  $\mathbf{X}_c \in \text{Partition}(\mathbf{X}_{\text{flat}}, C)$  do
4:   Retrieval
5:   Compute similarity matrix:  $\text{sim} \leftarrow \frac{\mathbf{X}_c \mathbf{K}^\top}{\|\mathbf{X}_c\| \|\mathbf{K}\|}$   $\{\text{sim} \in \mathbb{R}^{C \times M}\}$ 
6:    $\mathbf{S}^{\text{top}}, \mathbf{I}^{\text{top}} \leftarrow \text{TopK}(\text{sim}, k)$   $\{k \text{ nearest}\}$ 
7:    $\mathbf{S}^{\text{neg}}, \mathbf{I}^{\text{neg}} \leftarrow \text{TopK}(-\text{sim}, k')$   $\{k' \text{ negative}\}$ 
8:   Enhancement
9:   for  $i \leftarrow 1$  to  $C$  do
10:     $\mu^{\text{pos}} \leftarrow \frac{1}{k} \sum_{j=1}^k \mathbf{V}[\mathbf{I}^{\text{top}}[i, j]]$ 
11:     $\mu^{\text{neg}} \leftarrow \frac{1}{k'} \sum_{j=1}^{k'} \mathbf{V}[\mathbf{I}^{\text{neg}}[i, j]]$ 
12:    if  $\max(\mathbf{S}^{\text{top}}[i, :]) > \theta_{\text{high}}$  then
13:       $\mathbf{o}_i \leftarrow \mu^{\text{pos}} + \lambda \mu^{\text{neg}}$ 
14:    else if  $\theta_{\text{low}} < \max(\mathbf{S}^{\text{top}}[i, :]) \leq \theta_{\text{high}}$  then
15:       $\mathbf{o}_i \leftarrow \text{Avg}(\mu^{\text{pos}}, \mathbf{X}_c[i]) + \lambda \mu^{\text{neg}}$ 
16:    else
17:       $\mathbf{o}_i \leftarrow \mathbf{X}_c[i]$ 
18:    end if
19:     $\mathbf{O}_{\text{flat}}[i] \leftarrow \mathbf{o}_i$ 
20:  end for
21:  Update Phase
22:  Compute chunk mean:  $\mu_c \leftarrow \frac{1}{C} \sum_{i=1}^C \mathbf{X}_c[i]$ 
23:   $\mathbf{S}^{\text{topk}}, \mathbf{I}^{\text{topk}} \leftarrow \text{TopK}(\text{sim}, k)$ 
24:  if  $\frac{1}{k} \sum_{j=1}^k \mathbf{S}^{\text{topk}} > \theta_{\text{high}}$  then
25:    Update usage:  $\mathbf{u}[\mathbf{I}^{\text{topk}}] \leftarrow \mathbf{u}[\mathbf{I}^{\text{topk}}] + 1$ 
26:  else if  $\theta_{\text{low}} < \frac{1}{k} \sum_{j=1}^k \mathbf{S}^{\text{topk}} \leq \theta_{\text{high}}$  then
27:     $\mathbf{K}[\mathbf{I}^{\text{topk}}] \leftarrow \text{Avg}(\mathbf{K}[\mathbf{I}^{\text{top5}}], \mu_c)$ 
28:     $\mathbf{V}[\mathbf{I}^{\text{topk}}] \leftarrow \text{Avg}(\mathbf{V}[\mathbf{I}^{\text{top5}}], \mu_c)$ 
29:  else
30:    Find LRU slots:  $j^* \leftarrow \text{argmin}(\mathbf{u})$ 
31:    Replace:  $\mathbf{K}[j^*] \leftarrow \mathbf{X}_c, \mathbf{V}[j^*] \leftarrow \mathbf{O}_{\text{flat}}$ 
32:  end if
33: end for
34:  $\mathbf{O} \leftarrow \text{Reshape}(\mathbf{O}_{\text{flat}}, B, L, d_{\text{ff}})$ 
35: return  $\mathbf{O}, \{\mathbf{K}, \mathbf{V}, \mathbf{u}\}$ 
```

- **Output:** Enhanced activations \mathbf{O} and updated memory bank.

This algorithm enables low-overhead, context-sensitive memory usage for LLMs, simulating short-term working memory consolidation and reuse mechanisms.

C.2 Cortical Expert Clustering (CE)

Algorithm 2 shows how PaceLLM leverages cortical-like modularity by clustering FFN neurons across layers into interpretable experts using a constrained KMeans method.

- **Input:** Pretrained model \mathcal{M} and target number of experts K .
- **Output:** Updated model \mathcal{M}' with clustered and reordered FFN weights.

Explanation of key steps:

1. For each layer, extract FFN weights $\mathbf{W}_1^{(l)}$ (input projection) and $\mathbf{W}_2^{(l)}$ (output projection).

Algorithm 2 Cortical Expert Clustering (CE)

Require: Pretrained model \mathcal{M} , Number of experts K

```
1: Initialize empty state dictionary  $\mathcal{S}$ 
2: for layer  $l \in \{1, \dots, L\}$  do
3:   Extract FFN weights  $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}$ 
4:   if cluster indices  $\pi^{(l)}$  not cached then
5:     Compute  $\pi^{(l)} \leftarrow \text{KMeansConstrained}(\mathbf{W}_1^{(l)}, K)$ 
6:     Cache  $\pi^{(l)}$  to disk
7:   end if
8:    $\mathbf{W}_1^{\text{new}} \leftarrow \text{Rearrange}(\mathbf{W}_1^{(l)}, \pi^{(l)})$ 
9:    $\mathbf{W}_2^{\text{new}} \leftarrow \text{Rearrange}(\mathbf{W}_2^{(l)}, \pi^{(l)})$ 
10:  Update  $\mathcal{S}$  with  $\mathbf{W}_1^{\text{new}}, \mathbf{W}_2^{\text{new}}$ 
11: end for
12: return Model with updated weights  $\mathcal{M}'$ 
```

2. If the clustering result $\pi^{(l)}$ is not cached, apply constrained KMeans to group neurons into K expert clusters. This ensures load balance and specialization.
3. Rearrange the weight matrices according to cluster assignments $\pi^{(l)}$, so that expert-based routing can be implemented efficiently during inference.
4. Update the model's weight state dictionary with the new clustered weights.

This modularization allows PaceLLM to activate specific "experts" during computation and aligns with the cognitive hypothesis of cortical column specialization.

D Detailed Explanation of KMeans-Constrained Clustering and LRU Update Strategy

D.1 KMeans and Constrained KMeans Clustering for Expert Partitioning

D.1.1 Standard KMeans Clustering

Given N data points $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^d$, KMeans aims to find K clusters $\{\mathcal{C}_k\}_{k=1}^K$ and centroids $\{\boldsymbol{\mu}_k\}_{k=1}^K$ minimizing the intra-cluster variance:

$$\min_{\{\mathcal{C}_k\}} \sum_{k=1}^K \sum_{\mathbf{x}_i \in \mathcal{C}_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^2, \quad \text{where } \boldsymbol{\mu}_k = \frac{1}{|\mathcal{C}_k|} \sum_{\mathbf{x}_i \in \mathcal{C}_k} \mathbf{x}_i. \quad (12)$$

Iterative procedure:

$$\text{Assignment: } \mathcal{C}_k \leftarrow \left\{ \mathbf{x}_i : k = \arg \min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|_2 \right\} \quad (13)$$

$$\text{Update: } \boldsymbol{\mu}_k \leftarrow \frac{1}{|\mathcal{C}_k|} \sum_{\mathbf{x}_i \in \mathcal{C}_k} \mathbf{x}_i \quad (14)$$

Repeat until convergence.

D.1.2 Constrained KMeans Clustering

To prevent cluster imbalance, we impose cardinality constraints:

$$L_{\min} \leq |\mathcal{C}_k| \leq L_{\max}, \quad \forall k \in \{1, \dots, K\} \quad (15)$$

Special cases:

- **Equal-size constraint:** $|\mathcal{C}_k| = \frac{N}{K}$
- **Upper-bound constraint:** $|\mathcal{C}_k| \leq U$

Heuristic optimization: Let $d_{ik} = \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2$. We define the cluster assignment function as:

$$\pi(i) = \arg \min_{k \in \mathcal{A}_i} d_{ik}, \quad \mathcal{A}_i = \{k : |\mathcal{C}_k| < L_{\max}\} \quad (16)$$

That is, each \mathbf{x}_i is assigned to the nearest cluster among those with remaining capacity.

D.1.3 Application in PaceLLM

In FFN layers, each neuron corresponds to a row $\mathbf{w}_i \in \mathbb{R}^{d_{\text{model}}}$ of the weight matrix $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$. To enable sparse expert routing, we perform constrained clustering:

$$\{\mathbf{w}_i\}_{i=1}^{d_{\text{ff}}} \xrightarrow{\text{Constrained KMeans}} \{\mathcal{E}_k\}_{k=1}^K, \quad \text{where } |\mathcal{E}_k| = \frac{d_{\text{ff}}}{K} \quad (17)$$

Each expert \mathcal{E}_k serves as a functional block activated conditionally during inference.

Why clustering in PaceLLM?

- Reduces redundant neuron computation via routing.
- Ensures fair expert load balancing, avoiding expert collapse.
- Enables structure-aware specialization, as neurons with similar semantic roles are grouped.

D.2 Least Recently Used (LRU) Update Strategy for Memory Management

D.2.1 Mathematical Formulation

Let memory bank $\mathcal{M} = \{(\mathbf{k}_i, \mathbf{v}_i, u_i)\}_{i=1}^M$ store key-value pairs and their usage counters. At each time step t :

$$u_i(t) = \begin{cases} 0, & \text{if slot } i \text{ is accessed} \\ u_i(t-1) + 1, & \text{otherwise} \end{cases} \quad (18)$$

When writing a new memory $(\mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}})$, we check similarity:

$$\max_i \text{sim}(\mathbf{k}_{\text{new}}, \mathbf{k}_i) < \theta_{\text{low}} \Rightarrow \text{need replacement} \quad (19)$$

We replace the least recently used slot:

$$i^* = \arg \max_i u_i, \quad (\mathbf{k}_{i^*}, \mathbf{v}_{i^*}) \leftarrow (\mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}}), \quad u_{i^*} \leftarrow 0 \quad (20)$$

D.2.2 Application in PaceLLM

To model human-like memory with decay, PaceLLM uses a bounded-size memory \mathcal{M} and LRU strategy for updates:

- Prevents unbounded memory growth.
- Automatically decays outdated context.
- Encourages dynamic adaptation to new content.

Why LRU in PaceLLM?

- Emulates neural memory fading (forgetting).
- Reduces retrieval noise by replacing stale keys.

- Aligns with human working memory dynamics, where recent tokens dominate attention.

Together, constrained KMeans and LRU form the foundation of PaceLLM’s architecture:

Expert Routing + Working Memory Adaptation \Rightarrow **Efficient and Continual Inference**

E Extra Experiments on More Models

Table 8: Performance comparison between PaceLLM and Mistral-7B-Instruct-v0.3 on LongBench tasks in **training-free** manner. CE denotes cortical expert neuron clustering and PA means persistent activity memory mechanism.

Method	Single-Document QA				Multi-Document QA				Summarization				Few-shot Learning				Code	
	NtyQA	Qasper	MFen	Avg.	HopaQA	2WikiMQA	Musique	Avg.	GovReport	OMSum	MultiViews	Avg.	TREC	TriviaQA	SAMSum	Avg.	Loc	RB-P
Mistral	Vanilla	29.82	41.12	53.75	41.56	49.87	39.51	28.34	39.24	35.88	25.55	27.85	29.76	76.0	88.89	47.32	70.74	59.20
	Vanilla+CE	26.15	43.46	45.45	38.35	55.91	56.31	35.42	49.21	35.53	23.44	26.78	28.58	78.0	89.41	45.03	70.81	53.76
	Vanilla+PA	29.30	41.16	53.76	41.41	50.61	39.84	28.96	39.80	36.40	25.64	27.27	29.77	76.0	89.56	47.27	70.94	59.74
	Vanilla+PA+CE	30.10	43.68	54.06	42.61	56.97	56.31	35.63	49.64	36.63	26.65	27.63	30.30	78.0	89.41	48.03	71.81	59.76
																	60.85	60.30
																	60.89	60.33

Table 9: Performance comparison of more baseline models and our method (CE + PA) on LongBench, aggregated into major task categories. Results show consistent improvements across architectures in a **training-free** manner.

Model	SQA	MQA	Sum.	FSL	Cod.
Qwen2.5-14B-Instruct	17.18	12.15	23.35	71.46	32.30
Qwen2.5-14B-Instruct+Ours	18.48	12.97	23.49	72.32	33.41
Llama-3.1-8B-Instruct	24.22	15.04	28.21	69.49	58.44
Llama-3.1-8B-Instruct+Ours	24.31	15.80	28.47	69.85	59.59

Results on LongBench with Mistral. Table 8 reports the training-free evaluation results of the Mistral model across different LongBench tasks. We observe that both the cortical expert neuron clustering (CE) and persistent activity memory (PA) modules individually enhance the base Mistral model in different task categories.

Specifically, CE brings notable improvements in multi-document QA, with performance in 2WikiMQA and Musique boosted by up to 16.8% and 7.1% respectively compared to the vanilla model. This confirms CE’s effectiveness in capturing complex cross-document reasoning patterns. On the other hand, PA contributes consistently across all categories, particularly maintaining or even slightly improving the base performance in summarization and few-shot tasks, while preserving high accuracy in code reasoning.

When both mechanisms are combined (CE+PA), the model achieves the best overall results, outperforming the vanilla baseline in 13 out of 16 subtasks. Notably, the average accuracy in Single-Document QA improves from 41.56% to 42.61%, and in Multi-Document QA from 39.24% to 49.64%, representing a 10.4% absolute gain. Summarization and code tasks also benefit from the combination, indicating that the two brain-inspired components are complementary.

These results demonstrate that our proposed architecture not only generalizes well across task types but also significantly strengthens the model’s long-range reasoning capability in a fully training-free setting.

Results on LongBench with Qwen2.5 and Llama3.1. Table 9 presents the performance of our method when applied to two state-of-the-art LLMs — Qwen2.5-14B-Instruct and Llama-3.1-8B-Instruct — under the same training-free setup. Despite their different architectures and training corpora, both models exhibit consistent improvements across all task categories after integrating our brain-inspired mechanisms.

For Qwen2.5-14B-Instruct, the integration of CE and PA leads to gains in every domain, with particularly notable improvements in multi-document QA (+0.82) and code reasoning (+1.11). The

model also achieves higher accuracy in few-shot learning, suggesting that our memory mechanism enhances its ability to leverage contextual demonstrations without retraining.

Similarly, on Llama-3.1-8B-Instruct, our method consistently boosts performance across all five categories, even though the base model already performs strongly in code and single-document QA. The most significant gains occur in multi-document QA (+0.76) and summarization (+0.26), indicating that CE and PA help compensate for limitations in long-context integration, especially in models with smaller context windows or less optimized retrieval capabilities.

These results demonstrate that PaceLLM’s design is not only effective but also highly generalizable, delivering consistent benefits across diverse model families and scales. The fact that both a heavily optimized commercial-grade model (Qwen) and a compact open-weight model (Llama) benefit from our approach underscores its potential as a universal, plug-and-play enhancement for long-context understanding.