Saliency-Driven Dynamic Token Pruning (SDTP)

for Efficient Long-Context Inference

2025

- 本质就是不断减少 token，不是改词表、也不是删原始文本（输入的文本长度），而是在模型前向过程中，判断哪些 token 在后续层里不再重要，提前停止对它们的计算（内部计算时处理的 token 数量）
- LLM 训练的时候，用显著性分数给出 important or not important 的标注，冻结主模型，仅计算每层每个 token 的显著性（saliency）作为监督指标
- 训练一个轻量剪枝模块 用来做重要度预测，在每个 Transformer block 后接入一个小型 MLP，输入为该层隐藏状态，输出为每个 token 的重要度分数。
- 推理时用剪枝模块的分数对 token 排序，只保留一定比例（比如 90% / 80%），其余 token 不再进入下一层计算。为稳健，有一个强制的设定保留开头少量 token 和一小段局部上下文。
- 越往后，真正参与计算的 token 越少，自注意力的计算与显存占用随之下降

主要就是四个步骤

1. 模型与环境搭建 requirements.txt / env.yml
2. Saliency 提取与标注阶段
3. SDTP 模块实现与训练
4. 推理与评测实验

➢ **实验验证 // 如何判断“减少 token 但效果没变”的主要指标**

(1) 衡量 token 总体减少多少 Token Compression Ratio = 原始 token 数 ÷ 剪枝后 token 数
(2) 比较剪枝前后的前向计算速度 Prefill Speedup = 原始 prefill 时间 ÷ 剪枝后时间
(3) End-to-End Speedup 比较整体推理速度 = 原始总时间 ÷ 剪枝后总时间
(4) Memory Reduction 比较显存占用是否下降 = (原显存−剪枝显存) ÷ 原显存

(5) 性能 1, Task Accuracy / F1 评估任务表现是否下降，在 BoolQ / PIQA 任务上跑准确率
(6) 性能 2, Perplexity (PPL) 判断语言流畅性是否变化 = 剪枝后 vs 原模型 PPL 差异 < 5% 即为无损
(7) 性能 3, Loss Difference 训练后语言模型损失变化 = 剪枝模型 loss − 原模型 loss
(8) 性能 4, LongBench Score 的官方脚本测 长上下文理解性能

以 llama2-7b 为主要模型，完成后再考虑 Qwen2-7B、Gemma-7B、gpt-oss-20b

Note ：减少 token 可以指训练成本中 token 的减少，也可以指使用 llm (API 计费) 时 token 的减少，其中前者是这篇，后者是和 tokenizer 相关的 用户输入端的上下文压缩

1. 输入 token 数（Input Token Count）：由 tokenizer（通常是 SentencePiece/BPE）分词后得到的输入长度。

2. 层内有效 token 数（Active Tokens per Layer）：每一层中经过剪枝后仍然参与计算的 token 数。用来计算 “Compression Ratio” 和 “Speedup” 的核心指标

指标	数据来源	计算方式	工具 / 实现
原始 token 数	Tokenizer 输出长度	len(input_ids)	Transformers tokenizer
每层保留 token 数	SDTP mask 输出	mask.sum()	PyTorch forward hook
平均压缩比	各层 token 数平均值	mean(N_kept_per_layer) / N_input	Python aggregation
Prefill 时间	GPU 计时	torch.cuda.Event	PyTorch
显存占用	最大显存分配	torch.cuda.max_memory_allocated()	PyTorch CUDA 工具

1. Background and Motivation

❖ LLMs experience a rapid increase in computation and memory cost as input length grows.

❖ While prompt compression and KV cache reduction focus on input and output efficiency, the intermediate token redundancy inside model layers is largely overlooked.

❖ The paper *“Saliency-Driven Dynamic Token Pruning (SDTP)”* propose a simple yet effective way to prune unimportant tokens during the prefill stage by learning token saliency.

❖ This project aims to reproduce the SDTP framework, evaluate its claimed efficiency improvements, and test its generalization to longer real-world contexts.

2. Objectives

➢ Reproduce SDTP on an open 7B model (Llama2 or Mistral).

➢ Verify speed-up and memory reduction without major accuracy loss.

➢ Provide a lightweight, reusable implementation compatible with HuggingFace Transformers.

➢ Explore a small extension: using attention entropy instead of gradient-based saliency as supervision.

3. Methodology

(1) Saliency Extraction

Compute gradient × input importance maps from frozen LLM layers to identify redundant tokens. Store normalized per-layer saliency values as supervision signals.

(2) SDTP Module Training

Attach a small MLP to each transformer block (hidden → hidden/4 → 2). Train only these modules using: L = L_mse + L_rank + L_lm. where ranking loss preserves order of importance and LM loss maintains semantics.

(3) Dynamic Pruning at Inference

Starting from layer 4, retain 90% tokens per stage (plus the first 4 tokens and local 10% context). Measure prefill latency, end-to-end speed-up, and accuracy change.

3.2 Token Reduction Mechanism

In SDTP, the reduction of tokens does not refer to altering the vocabulary or removing words from the input text. Instead, it dynamically decides during inference which tokens within the model layers are no longer necessary for further computation and prunes them on the fly.

At training time, each token's importance is measured using a saliency score based on gradient attribution. For a given token representation x_i at a certain layer, its saliency is defined as:

$$\text{saliency}_i = \left|\frac{\partial L}{\partial x_i} \cdot x_i\right|$$

where L is the language modeling loss.

A high saliency value indicates that changes to this token would strongly affect the model's output, while a low value suggests redundancy.

A lightweight MLP-based pruning module is then trained-under supervision of these saliency scores-to predic token importance directly from hidden states, without computing gradients at inference time.

During forward propagation, tokens are ranked by predicted importance, and only the top- r proportion (e.g., 90%) are retained for the next layer, while the rest are skipped.

This process is repeated across several layers, forming a layer-wise cascading reduction in token count:

$N, 0.9N, 0.81N, 0.73N, \ldots$

The compression ratio is defined as: $\text{Compression Ratio} = \frac{N_{\text{original}}}{N_{\text{after pruning}}}$

For example, reducing from 1000 to 620 tokens corresponds to a ratio of $1.61\times$, effectively lowering attention FLOPs by nearly half.

This dynamic pruning substantially reduces prefill latency and memory consumption while preserving the model's semantic fidelity.

4. Execution Plan

Day Range	**Goal**	**Deliverables**

Day 1–3	Environment setup, baseline tests	Run Llama2/Mistral 7B inference on LongBench, record baseline latency on RTX 5090.
Day 4–6	Saliency extraction implementation	Compute and visualize per-layer saliency on sample prompts.
Day 7–9	SDTP module integration	Add MLP-based pruning blocks, build lightweight training scripts.
Day 10–12	Training and fine-tuning	Train SDTP modules on Dolly-15K with single 5090, adjust pruning ratio.
Day 13–14	Evaluation	Compare prefill/E2E speed and accuracy with baseline; verify claimed 1.3–1.7× acceleration.
Day 15	Reporting and reflection	Summarize results, analyze saliency stability, suggest next-step improvements.

5. Expected Outcomes

- Verified implementation of SDTP with measurable latency and memory gains.
- Reusable code package (sdtp_module.py, train_sdtp.py, evaluate.py).
- Experimental note comparing gradient-based vs. attention-based saliency.

6. Resources

- **Hardware:** 1× NVIDIA RTX 5090 (32gb).
- **Software:** PyTorch ≥ 2.1, Transformers ≥ 4.37, FlashAttention 2, Datasets.
- **Data:** Databricks Dolly-15K for training; LongBench for evaluation. (THUDM/LongBench)