LiteSys

Introduction

LiteSys is a lightweight, flexible serving system designed specifically for **academic research on large language models (LLMs)** (and implementation for Kinetics: Rethinking Test-Time Scaling Laws).

The goal of LiteSys is to offer a minimal yet efficient alternative to existing solutions:

- ⚡ **Faster than Hugging Face Transformers** for inference workloads, especially in batched decoding.
- **Easier to modify than vLLM or SGLang**, making it ideal for prototyping custom model architectures, attention mechanisms, and scheduling policies.

LiteSys emphasizes **modularity**, **hackability**, and **ease of debugging**, without sacrificing key features like continuous batching, KV cache management, and model-parallel support.

Whether you're working on novel LLM execution strategies, experimenting with sparse attention, or evaluating training artifacts—LiteSys gives you full control with minimal overhead.

Components

Model Executor

The **Model Executor** defines how LLM layers or modules are applied to inputs—essentially controlling the model's forward workflow. Model implementations are located in `litesys/models/`.

Currently supported models:

- `Qwen2.5`
- `Qwen3`
- `Qwen3-MoE`

These correspond to model classes:

- `"qwen2"`
- `"qwen3"`
- `"qwen3moe"`

To add a new model:

1. **Define layer parameters** in `litesys/models/<new_model>_layer.py`.

2. **Implement execution logic** in `litesys/models/<new_model>.py`.

3. **(Optional)** Add tensor-parallelism support in `litesys/models/<new_model>_dist.py`.

    > Recommended when model weights use more than 40% of GPU VRAM.

4. **Register the model** in `litesys/models/auto_model.py`.

KV Manager

The **KV Manager** handles both:

- **Prefill stage**: single-token, single-request input.
- **Decoding stage**: efficient batched generation.

Key features:

- Optimized for **Grouped Query Attention (GQA)**:
  - KV caches are loaded once per group.
- Exposes **raw attention logits** (not fused), enabling rapid experimentation with sparse attention ideas like:
  - Native Sparse Attention
  - Block Sparse Attention
  - Quest
  - StreamingLLM
  - Mixed-head sparse attention variants

This modular design supports flexible and efficient attention strategies. This is why we do not used paged attention/flash attention.

Scheduler

The **Scheduler** supports **continuous batching**, meaning:

- As soon as a request finishes, a new one is fetched to fill its slot.
- Maximizes hardware utilization during long-generation workloads.

Repetition Detection

To avoid degenerate repetition loops (common in model evaluation), the scheduler supports early stopping:

Parameter	Description
`repeat_check`	Enable or disable repetition detection
`repeat_check_window`	How many recent tokens to scan for repeated patterns (e.g. 1024)
`repeat_block_size`	Length of repeated chunks to look for (e.g. 64 tokens)

> A request is terminated early if the last `repeat_check_window` tokens contain a number of exactly repeated `repeat_block_size`-sized blocks (e.g., 16).

This is especially useful for evaluating newly trained or unstable models.

Citation

If you use **LiteSys** in your research, please consider citing:

```
@misc{sadhukhan2025kineticsrethinkingtesttimescaling,
      title={Kinetics: Rethinking Test-Time Scaling Laws},
      author={Ranajoy Sadhukhan and Zhuoming Chen and Haizhong Zheng and
Yang Zhou and Emma Strubell and Beidi Chen},
      year={2025},
      eprint={2506.05333},
      archivePrefix={arXiv},
      primaryClass={cs.LG},
      url={https://arxiv.org/abs/2506.05333},
}
```