Priority-Aware Preemptive Scheduling for Mixed-Priority Workloads in MoE Inference

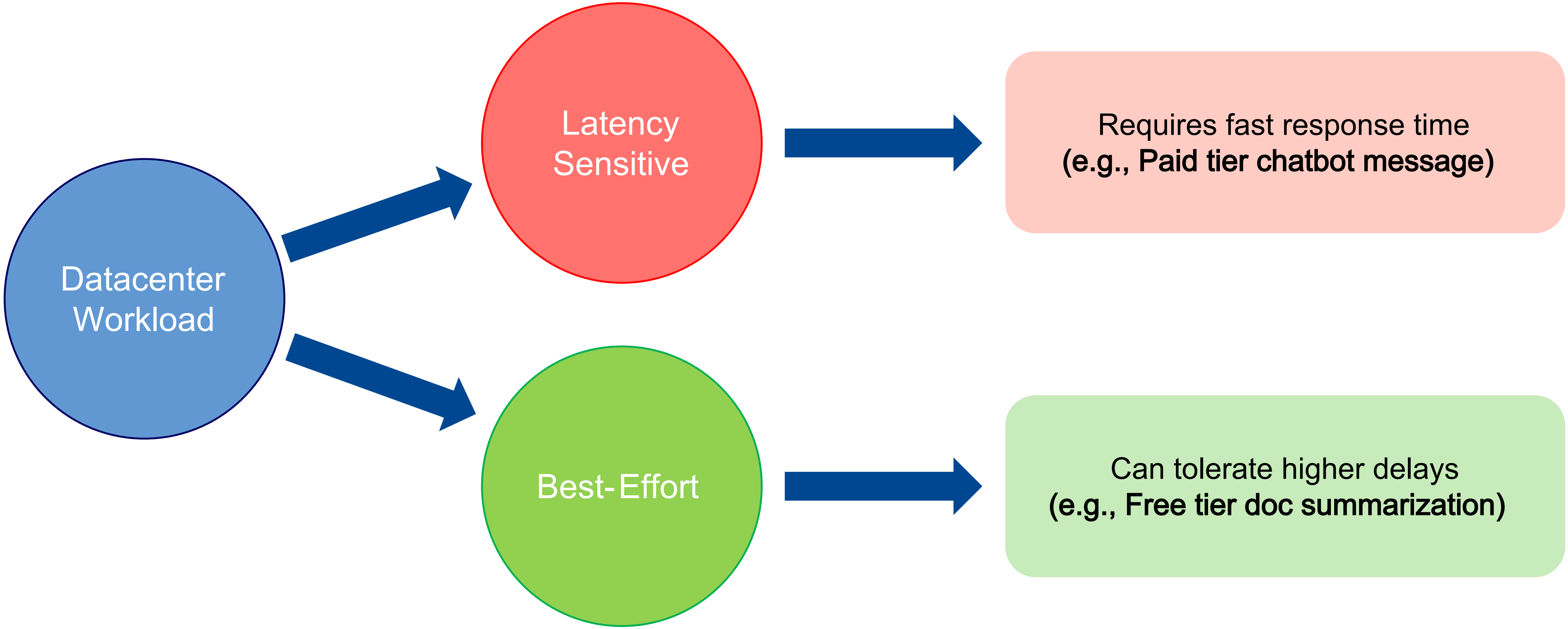
Mohammad Siavashi

Faezeh Keshmiri Dindarloo

Dejan Kostić

Marco Chiesa

Mixed-Priority Workload



Existing Systems Challenges

- First-Come-First-Served (**FCFS**) scheduling and **No priority differentiation** →Latency-Sensitive tasks wait behind Best-Effort tasks

- **Iteration -level scheduling** and **Run-to-Completion** batch execution

- **Long Best-Effort tasks** monopolize GPU resources →**Head of Line Blocking** (HOL)

What is the Solution?

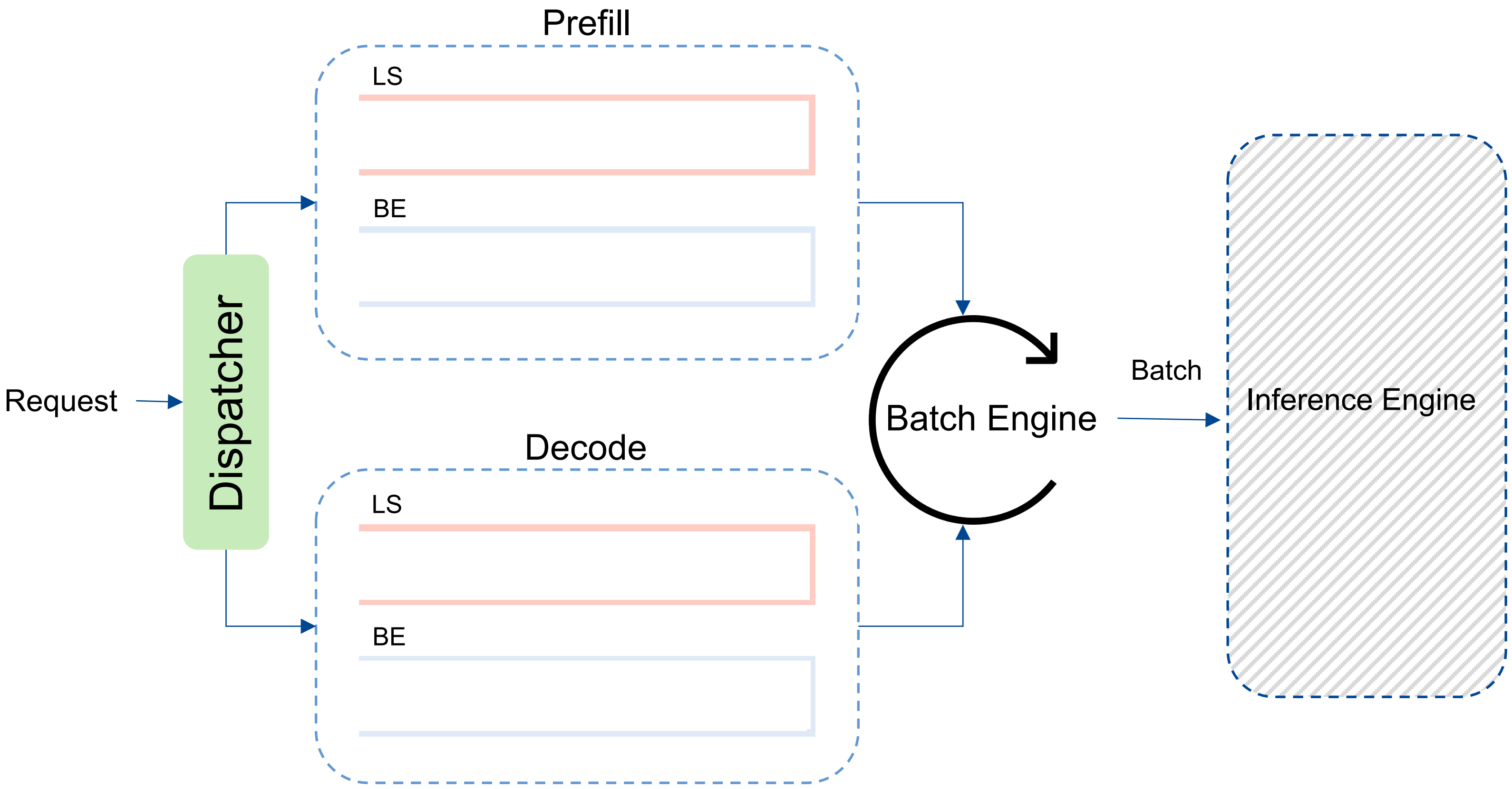
- A priority-aware scheduler → Requires preemption to be effective

- Challenges:
  - Models are oblivious to prompts/sequences internally → Models see only tensors
  - Caching state requires careful and expensive tensor operations → Delays preemption/resumption
  - Mixture of Experts models require additional effort in state management due to:
    - Dynamic routing
    - Top-k expert selection →Batch sequences must be tracked and synchronized over K experts

We design and implement QLLM with fast, fine-grained preemption and priority-aware scheduling.

QLLM Architecture Overview (Scheduler)

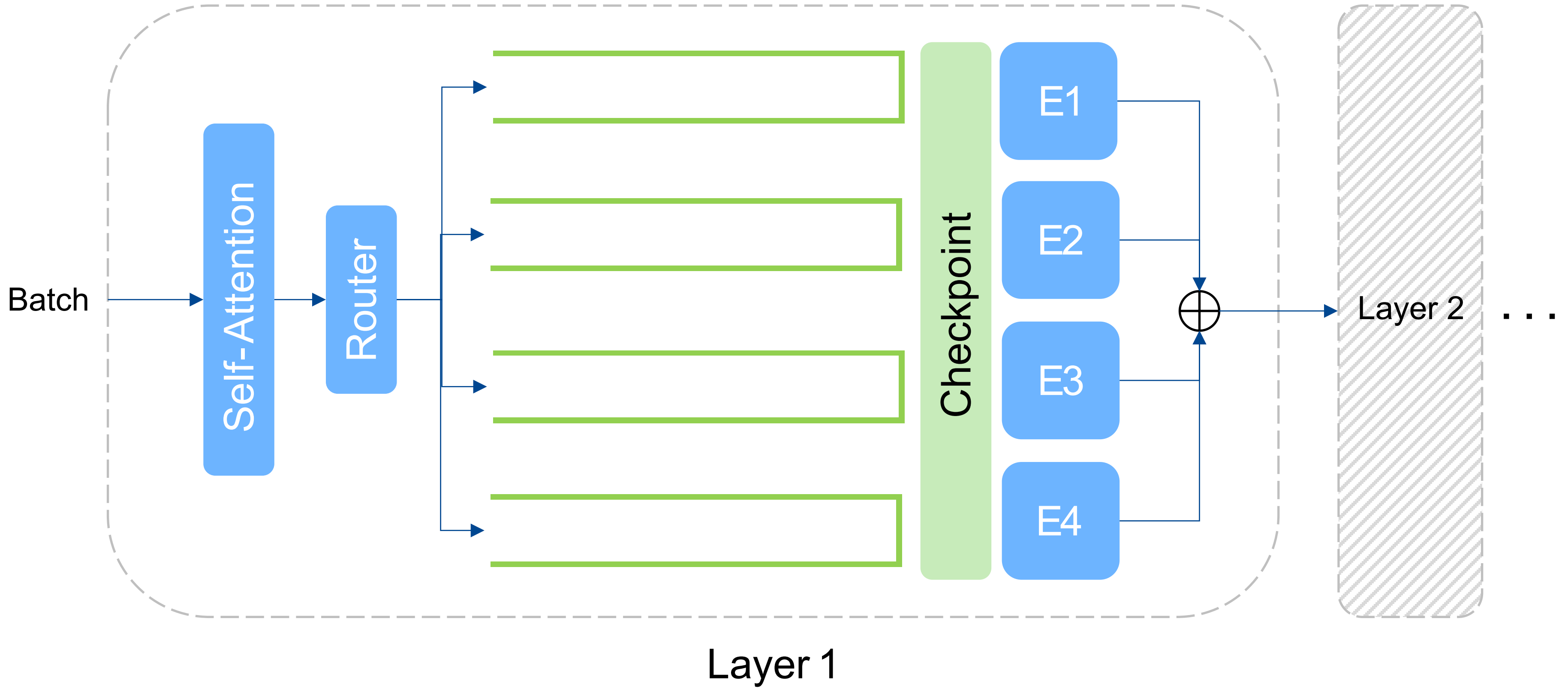
- Dispatcher distributes requests based on priority and phase
- Maintains Two Queues per Phase
- Implements Continuous Batching

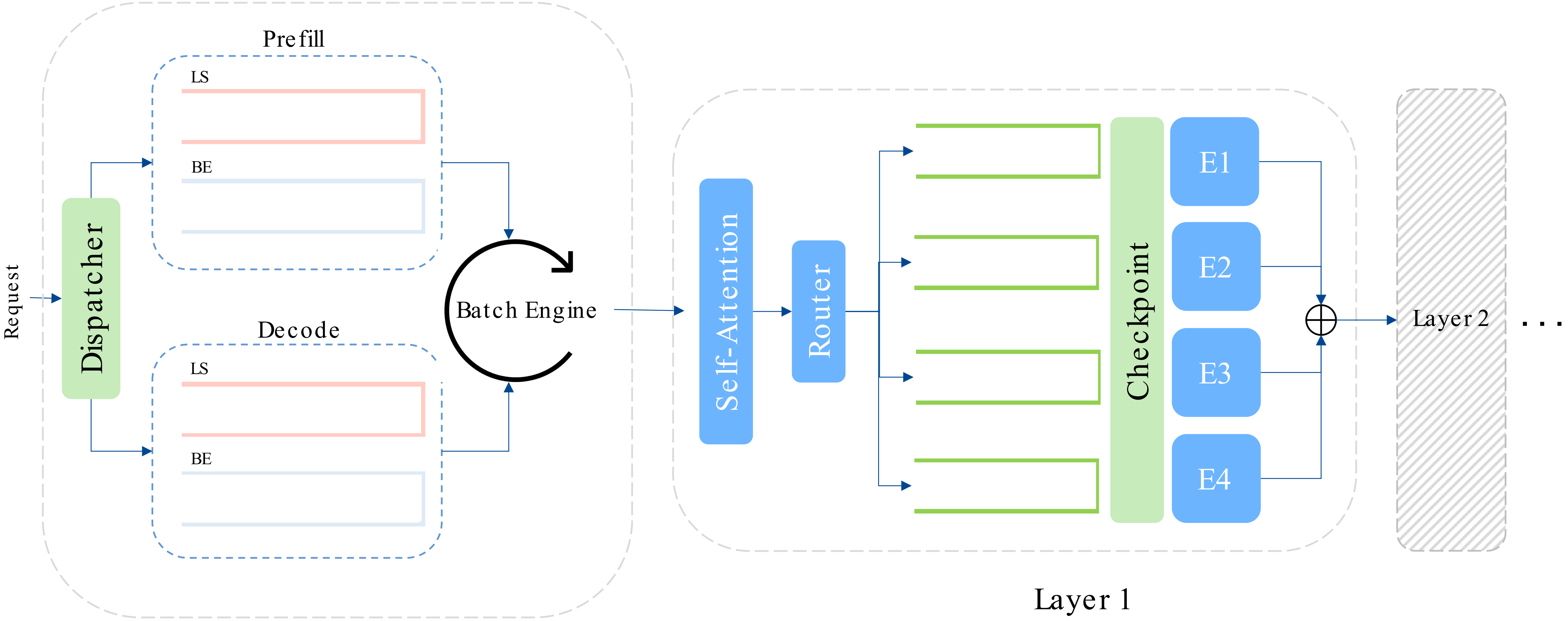


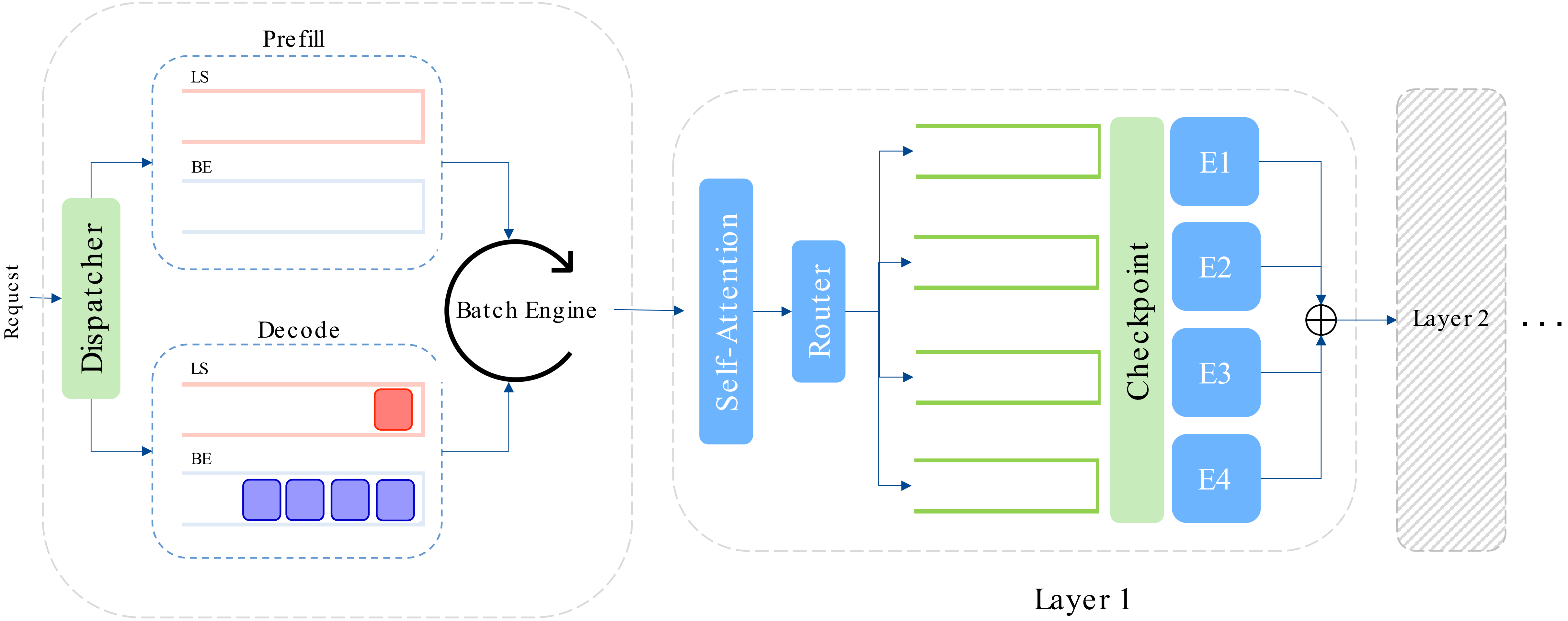
Refer to the paper for detailed scheduling policy.

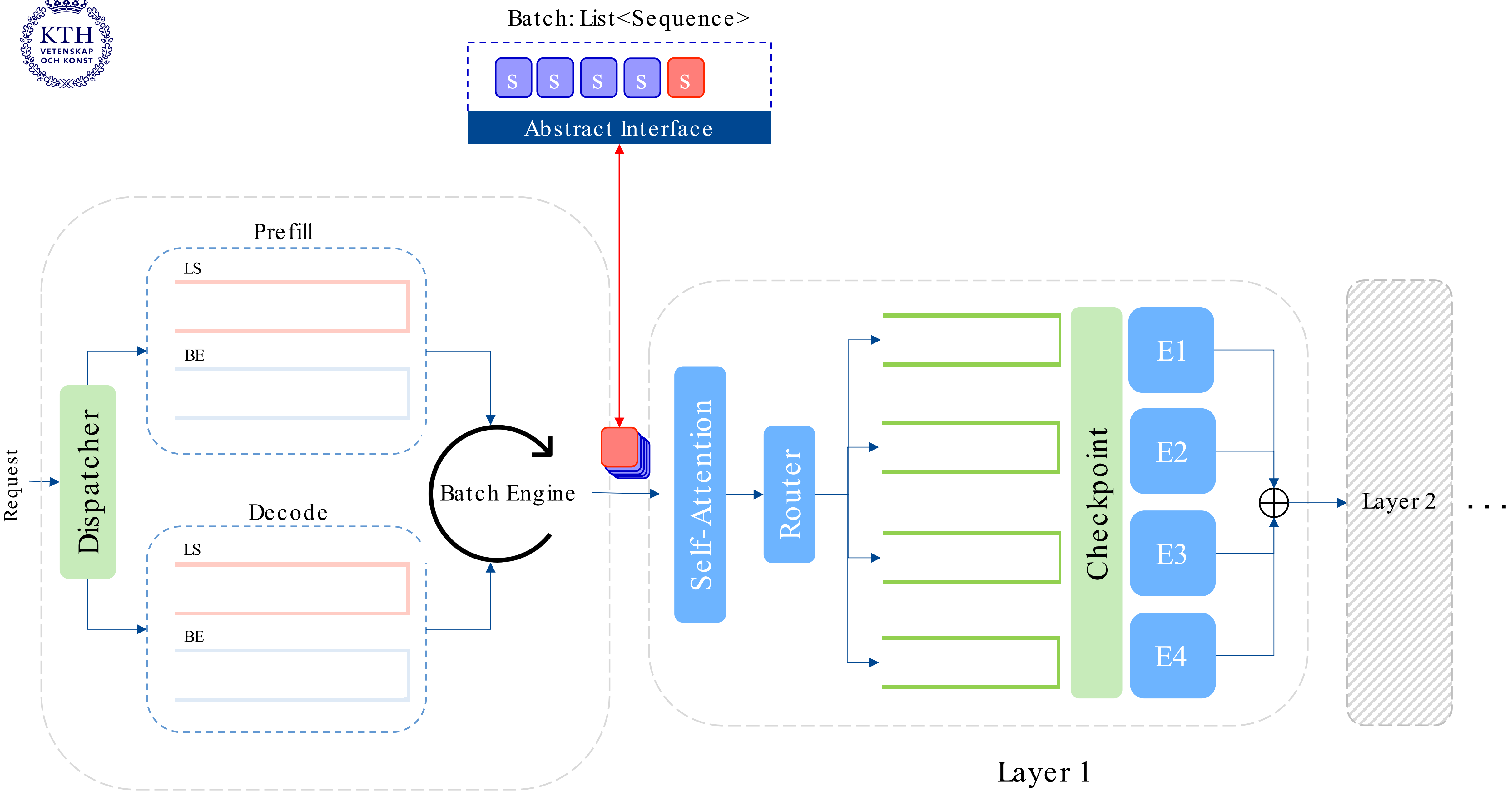
QLLM Architecture Overview (Inference Engine)

- New layer design with queues to flow -control
- Enables User-defined policy at each checkpoint through closed -loop controller mechanism

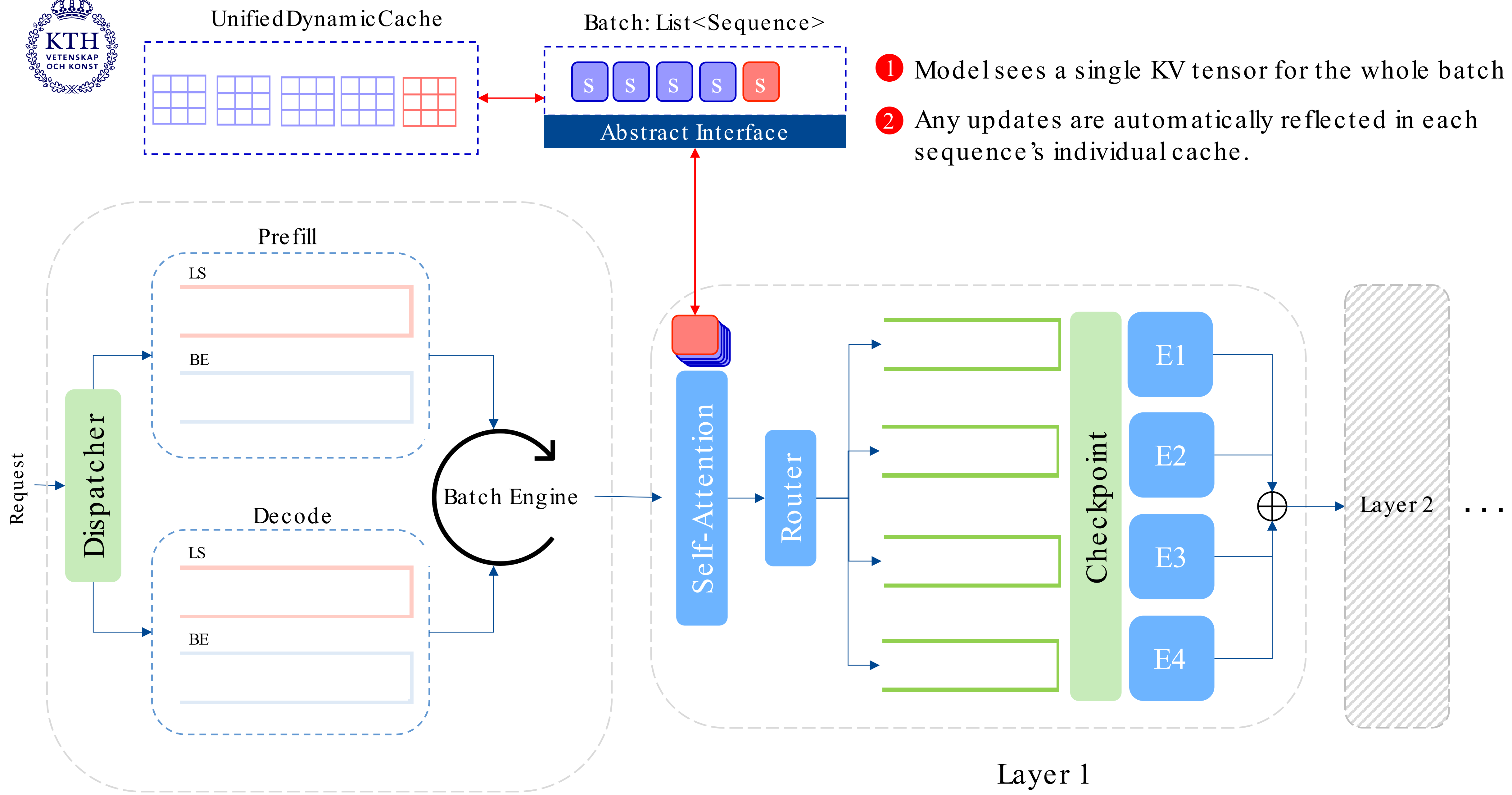


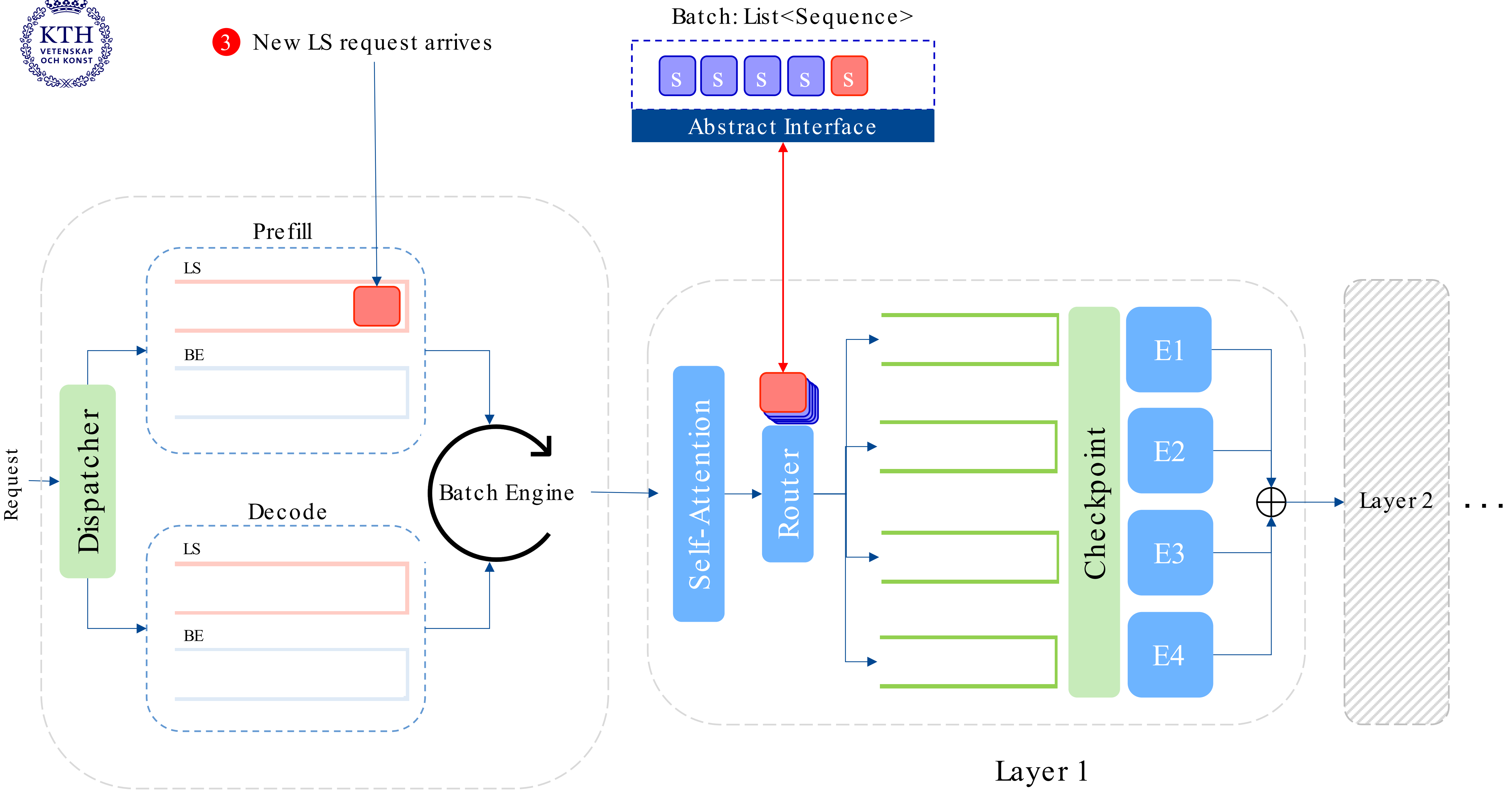

Request
Dispatcher
Prefill
LS
BE
Decode
LS
BE
Batch Engine
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1


Request
Dispatcher
Prefill
LS
BE
Decode
LS
BE
Batch Engine
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1

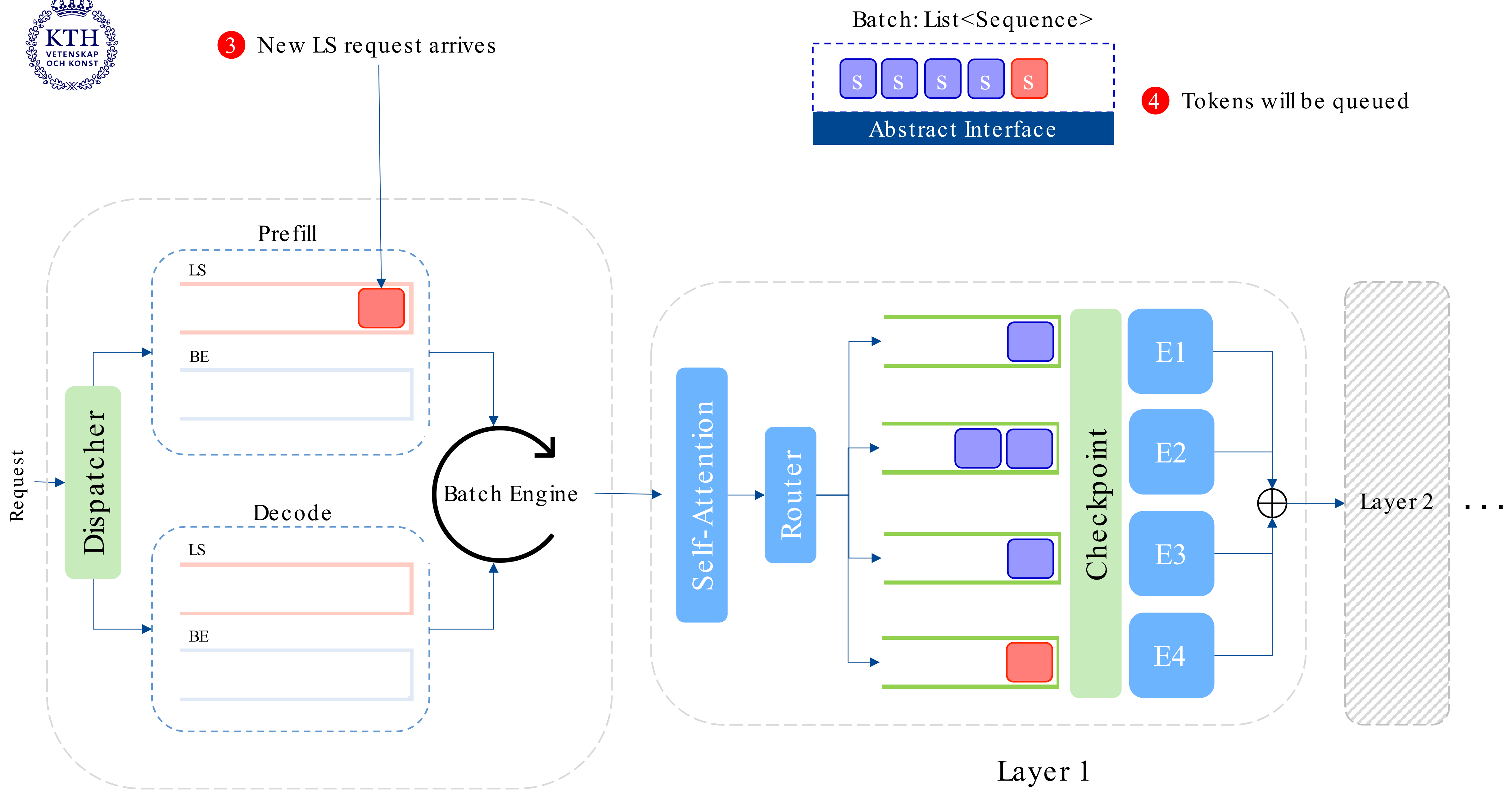


Batch: List<Sequence>
S
S
S
S
S
Abstract Interface
Prefill
LS
BE
Request
Dispatcher
Batch Engine
Decode
LS
BE
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1


UnifiedDynamicCache
Batch: List<Sequence>
S
S
S
S
S
Abstract Interface
1 Model sees a single KV tensor for the whole batch
2 Any updates are automatically reflected in each sequence's individual cache.
Prefill
LS
BE
Decode
LS
BE
Request
Dispatcher
Batch Engine
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1

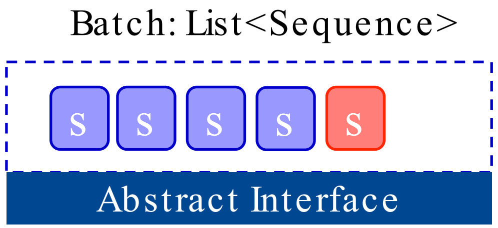


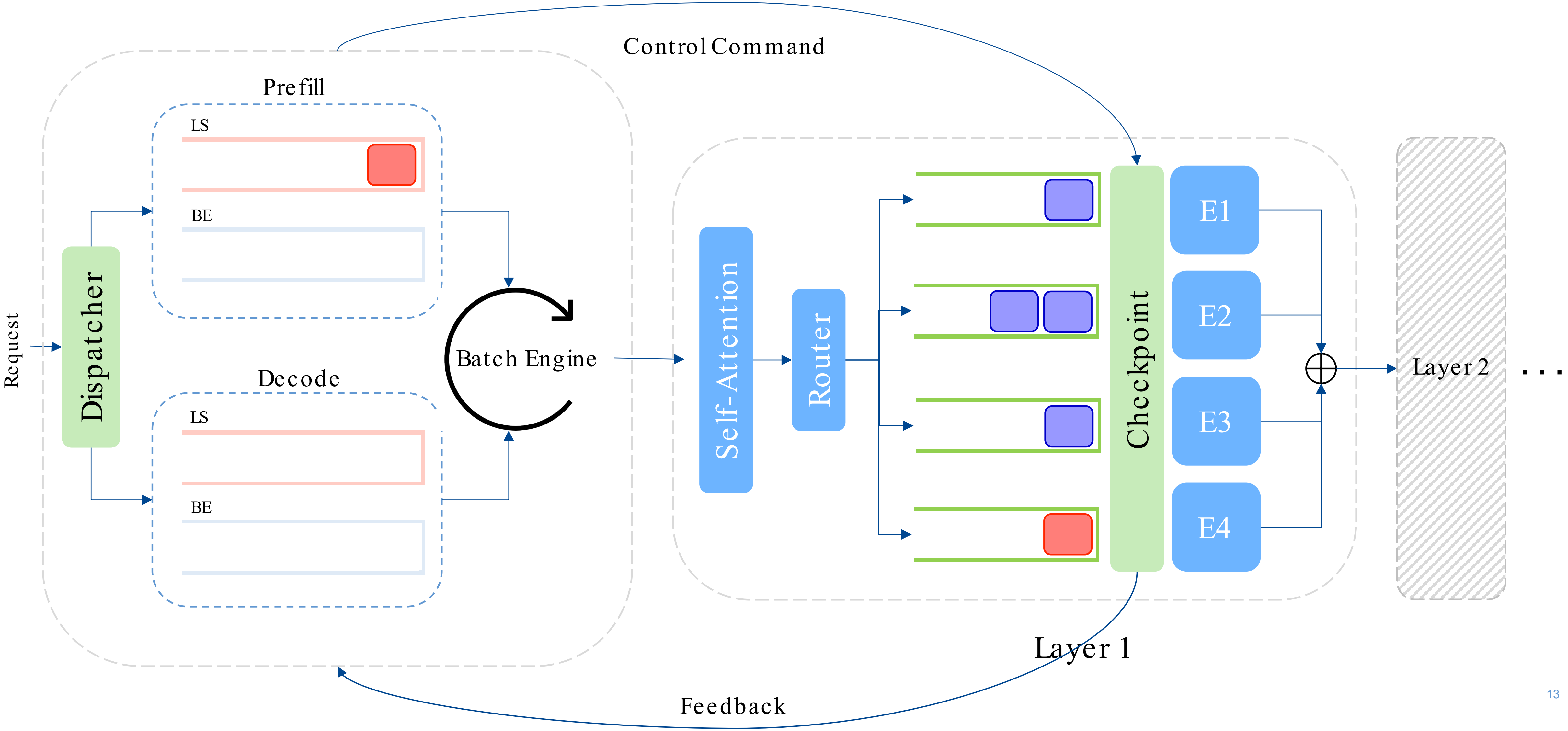
3 New LS request arrives
Batch: List<Sequence>
S
S
S
S
S
Abstract Interface
Prefill
LS
BE
Request
Dispatcher
Batch Engine
Decode
LS
BE
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1


Batch: List<Sequence>
S S S S S
Abstract Interface
3 New LS request arrives
4 Tokens will be queued
Prefill
LS
BE
Decode
LS
BE
Request
Dispatcher
Batch Engine
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1

5 Update the scheduler and receive control signal

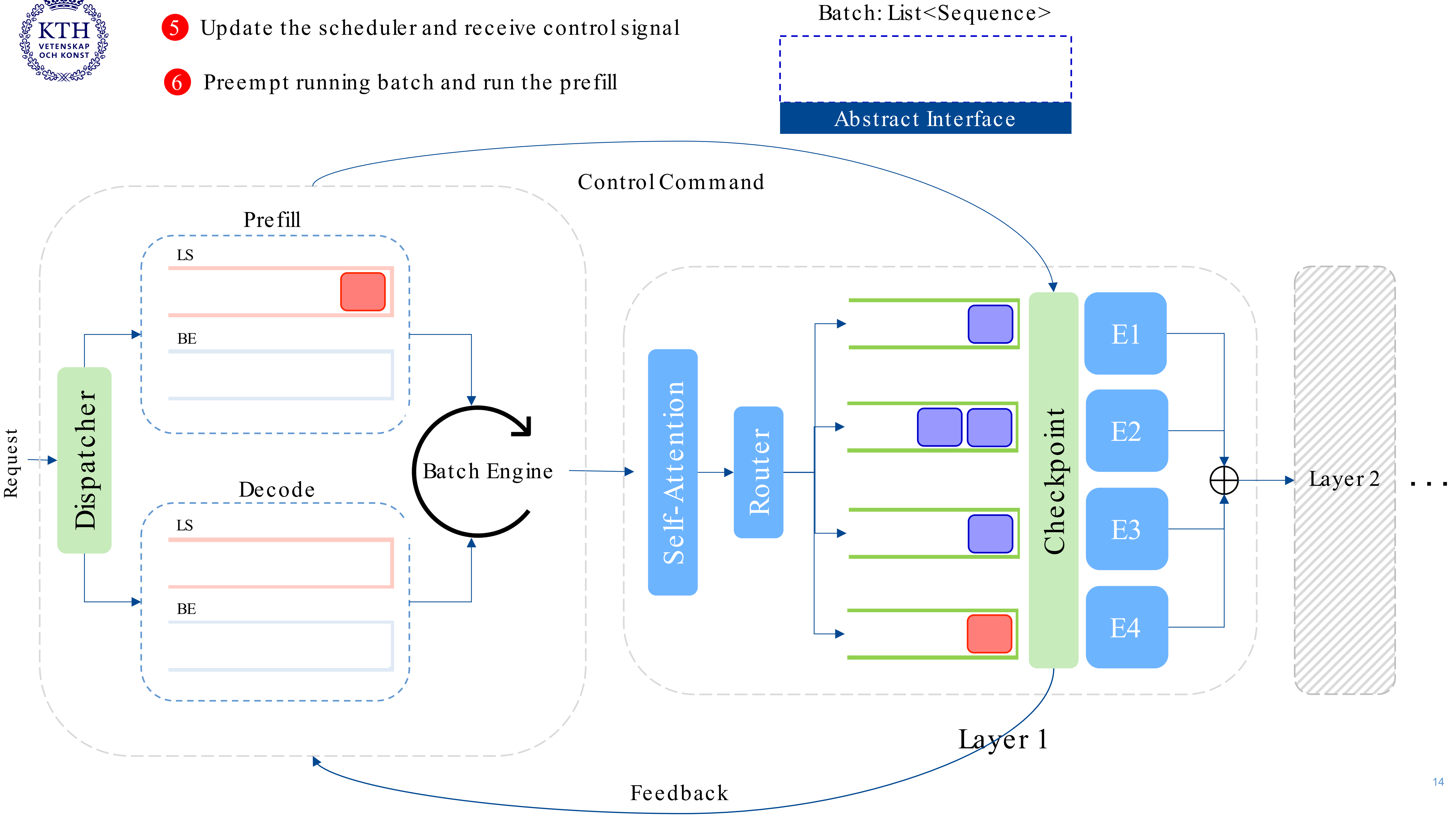
6 Preempt running batch and run the prefill





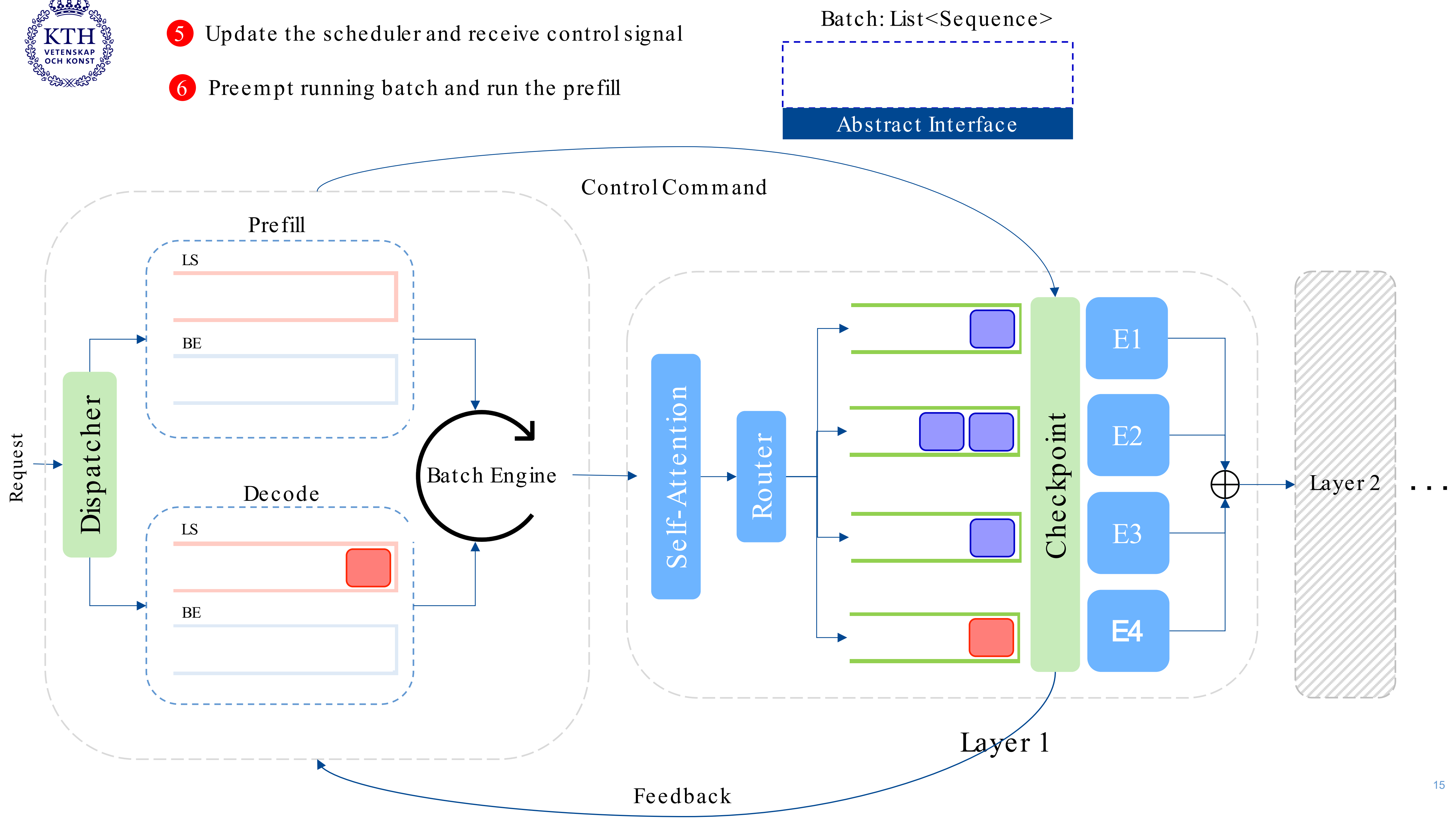
5 Update the scheduler and receive control signal

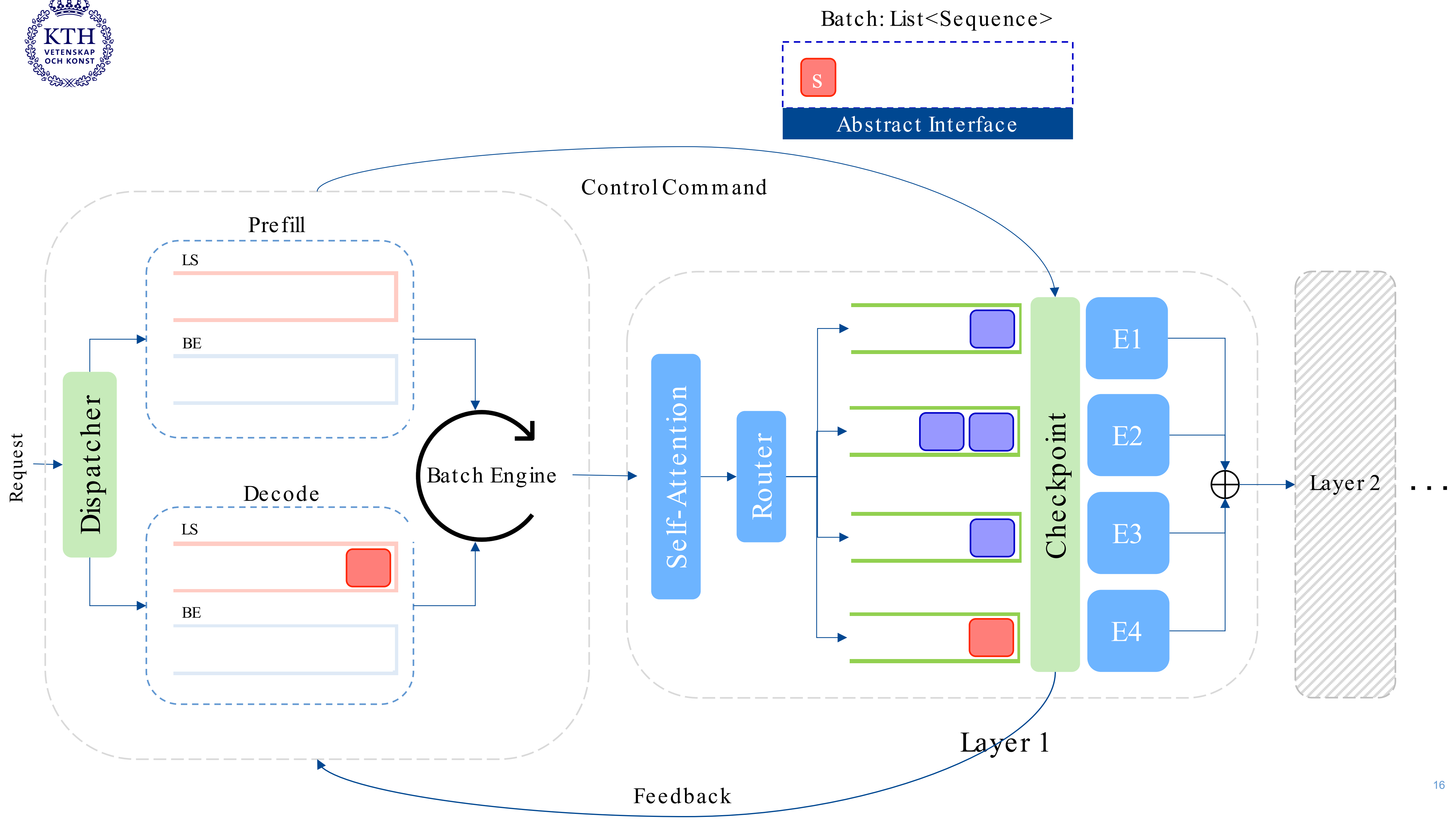
6 Preempt running batch and run the prefill



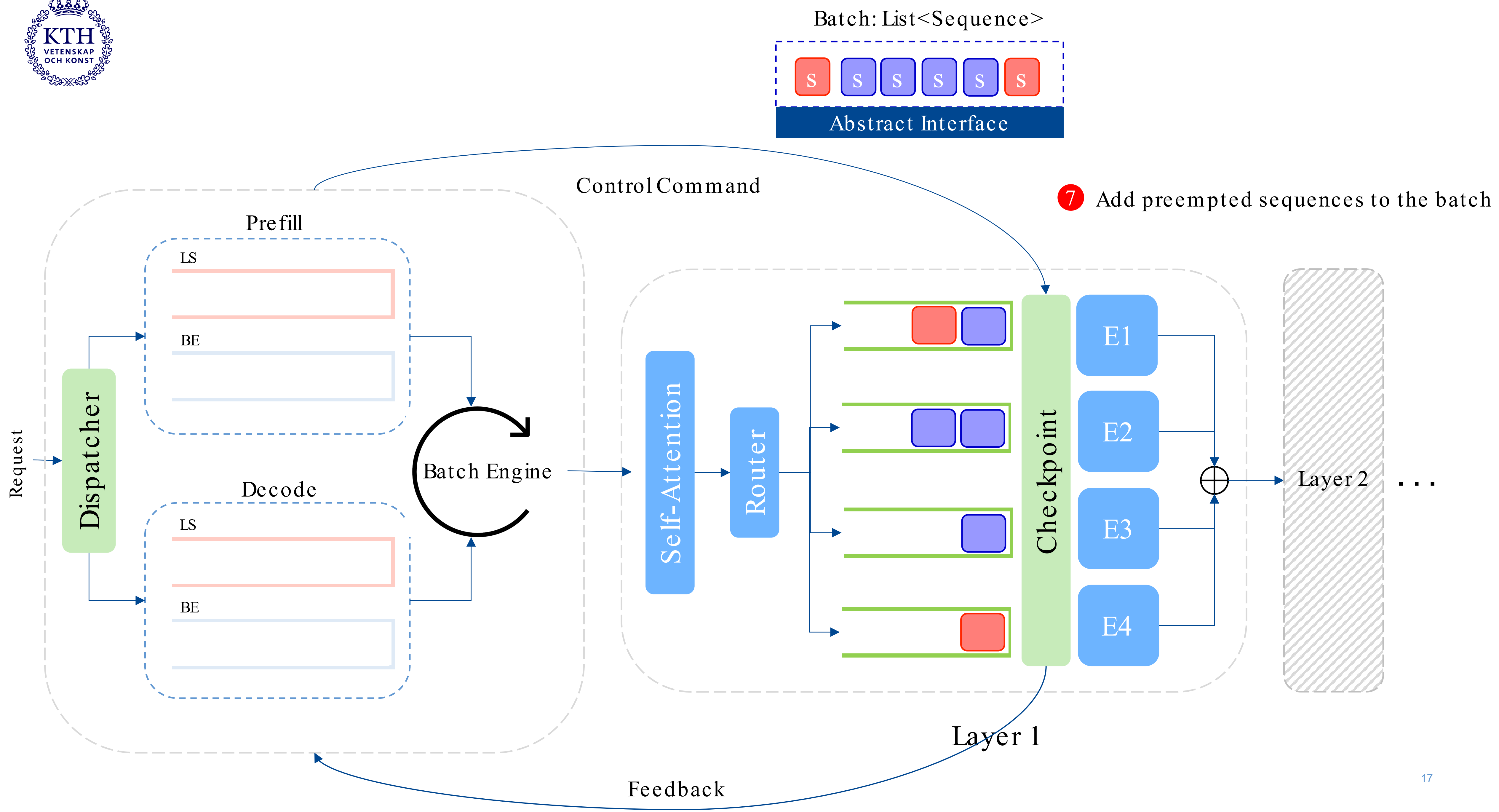
5 Update the scheduler and receive control signal

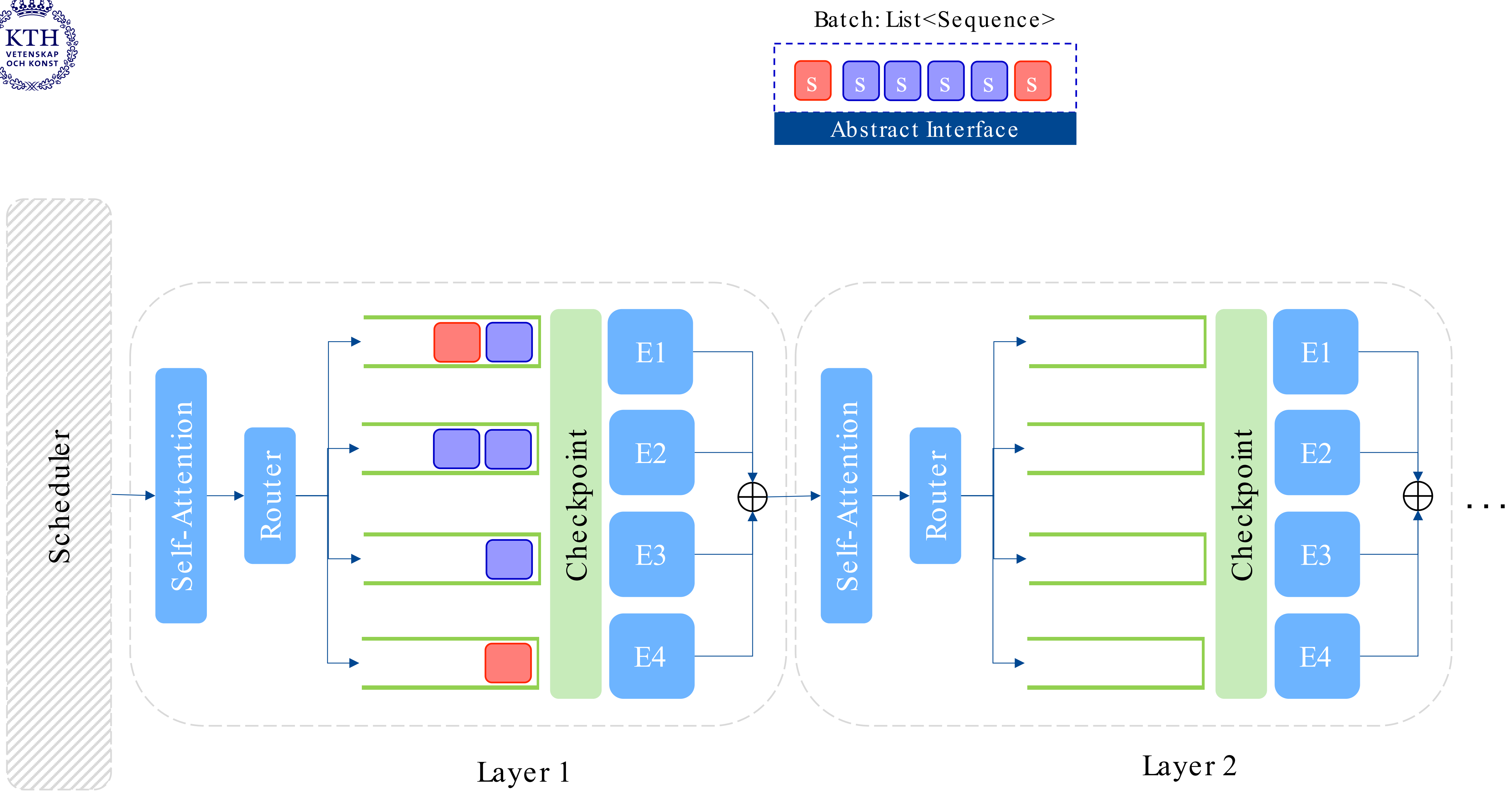
6 Preempt running batch and run the prefill



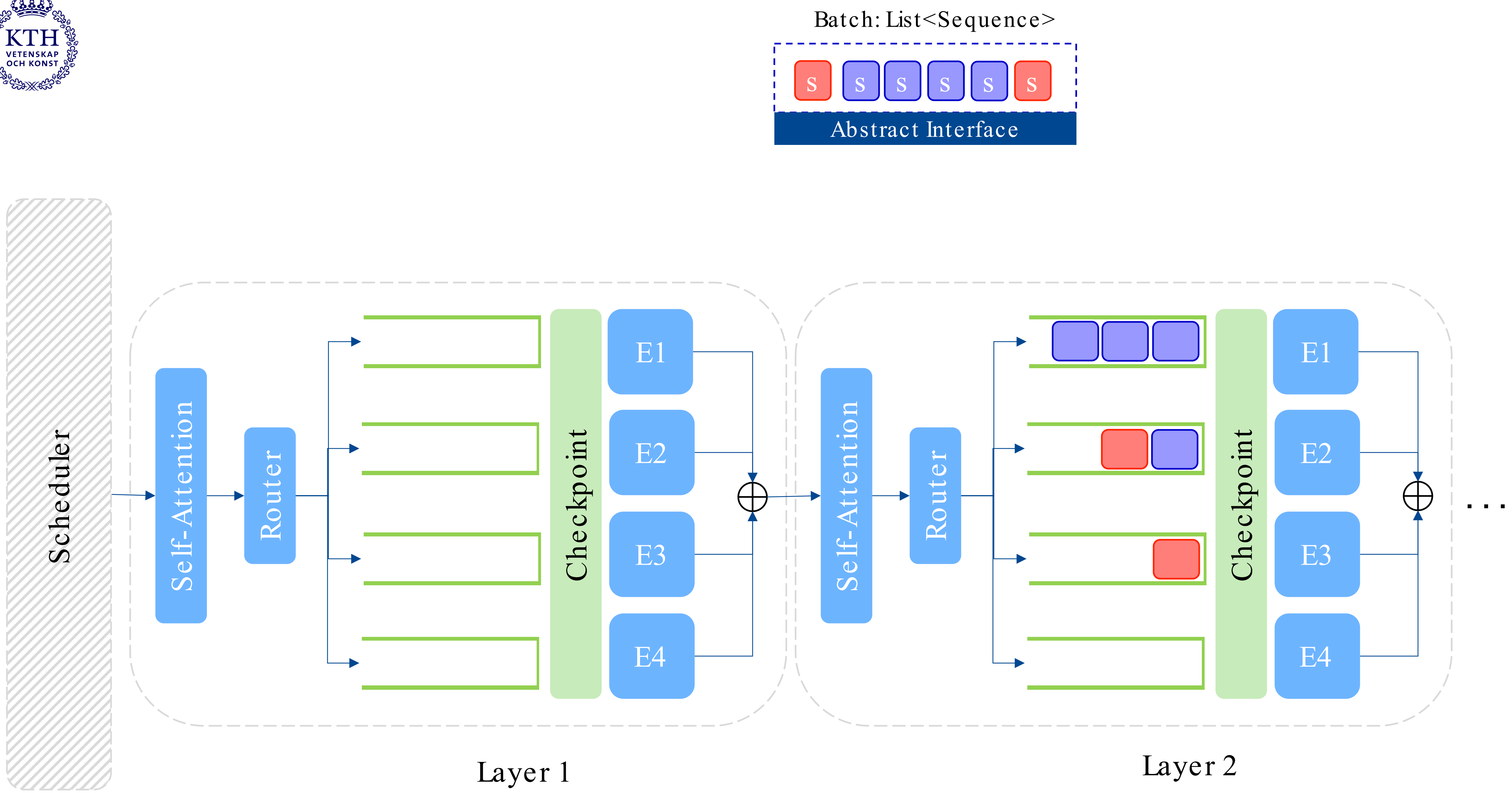


Batch: List<Sequence>
S
Abstract Interface
Control Command
Prefill
LS
BE
Request
Dispatcher
Batch Engine
Decode
LS
BE
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1
Feedback


Batch: List<Sequence>
S S S S S S
Abstract Interface
Control Command
7 Add preempted sequences to the batch
Prefill
LS
BE
Request
Dispatcher
Batch Engine
Decode
LS
BE
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .
Layer 1
Feedback



Batch: List<Sequence>
S
S
S
S
S
S
Abstract Interface
Scheduler
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 1
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .



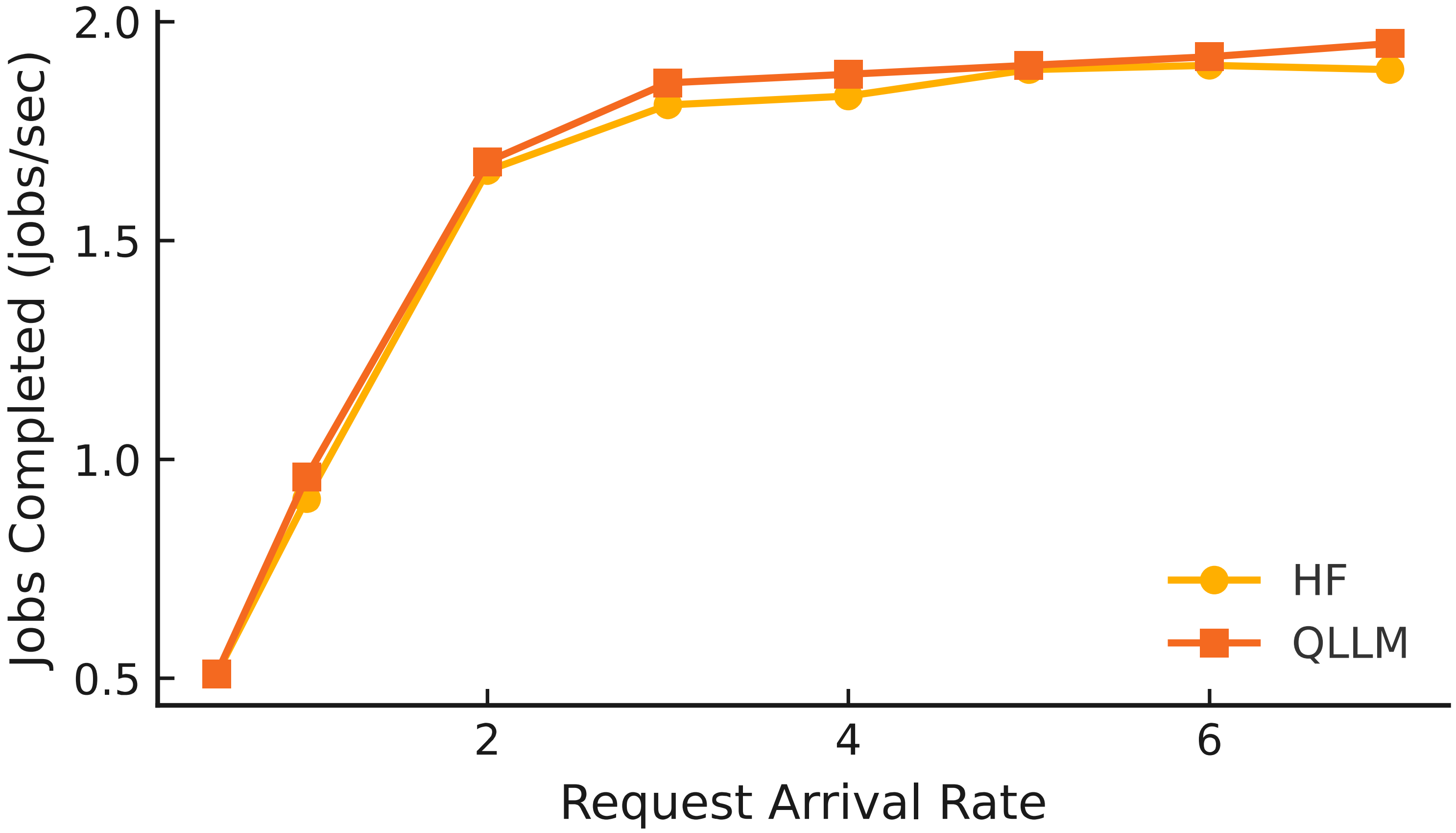
Batch: List<Sequence>
S
S
S
S
S
S
Abstract Interface
Scheduler
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 1
Self-Attention
Router
Checkpoint
E1
E2
E3
E4
Layer 2
. . .

Evaluation Setup

- Nvidia A100 80 GB GPU
- Dual-socket Intel Xeon Gold 6336Y CPU
- 256 GB DRAM

- Mixtral 8×7B in 4-bit quantization
- FP16 Precision
- ShareGPT dataset

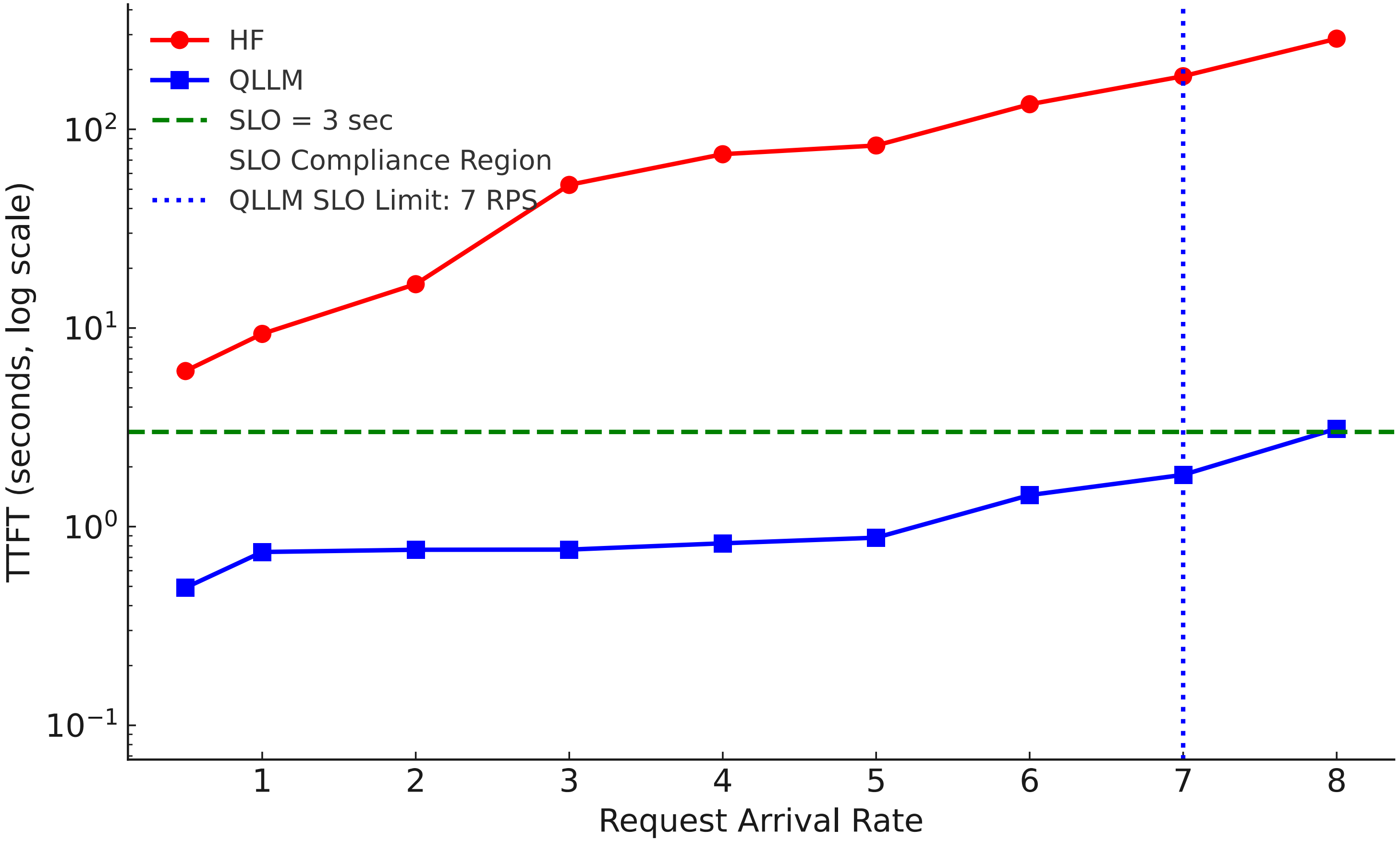
Throughput



2.0
1.5
1.0
0.5
Jobs Completed (jobs/sec)
2
4
6
Request Arrival Rate
HF
QLLM

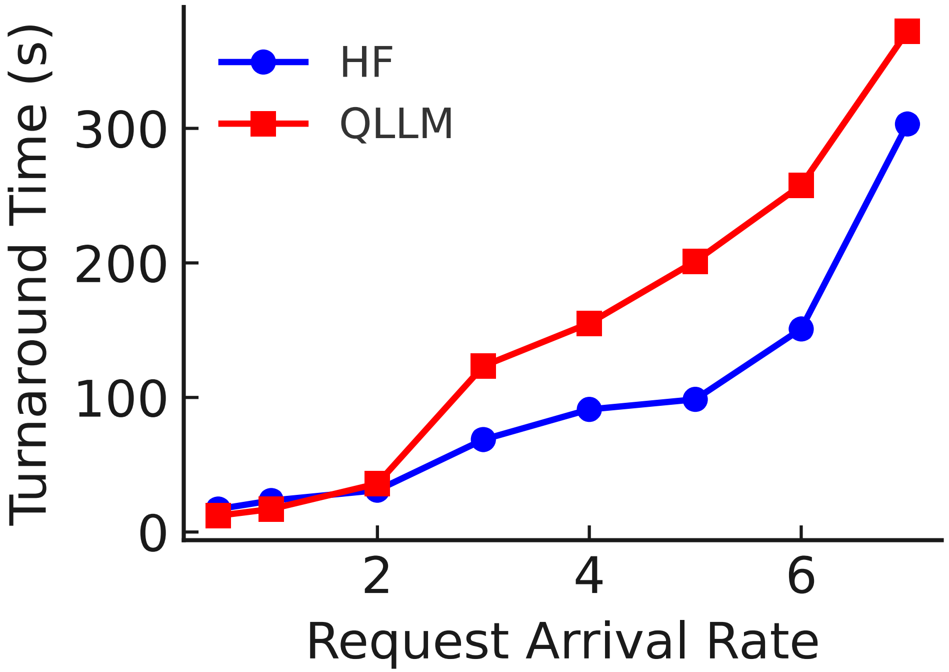
SLO Compliance for LS tasks

Up to **101.6x** improvement while ensuring compliance with the SLO (**Average of 65.2x**)

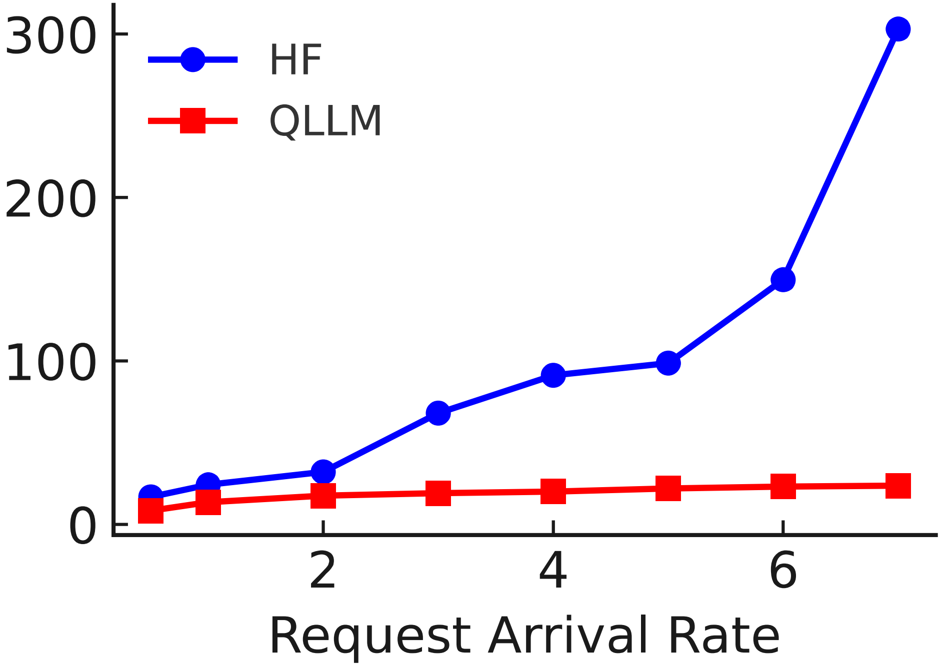


Turnaround Time

- Total time from when a request enters the system to when the full response is generated



Best-Effort



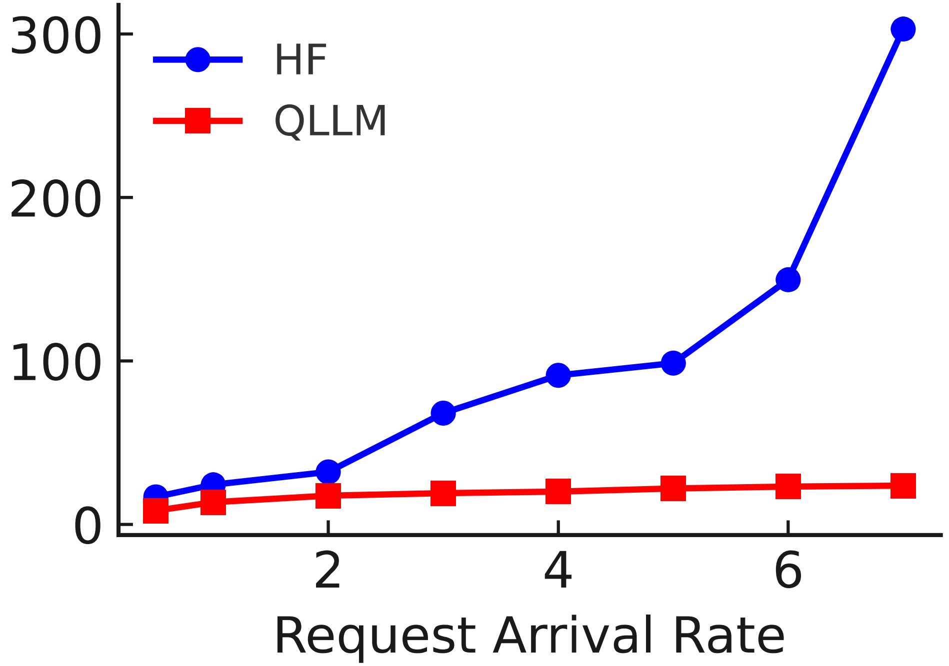
Latency-Sensitive

Turnaround Time

- Total time from when a request enters the system to when the full response is generated

Up to 12.8x

Best-Effort

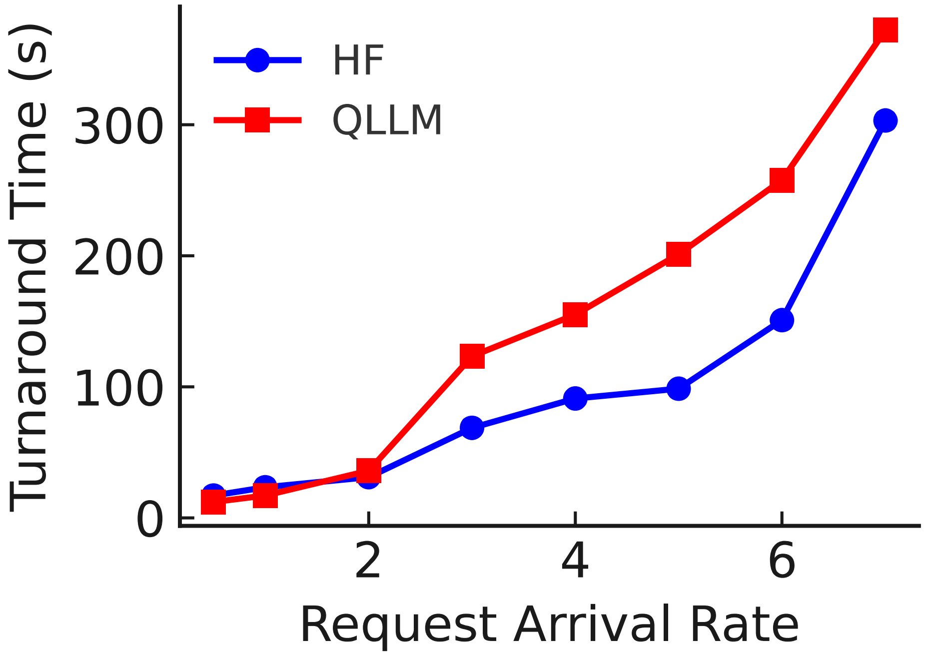


Latency-Sensitive

Turnaround Time

- Total time from when a request enters the system to when the full response is generated

1.38x Avg – 2.04x Peak



Best-Effort

Up to 12.8x

Latency-Sensitive

Conclusion

- **QLLM** introduces fine-grained, **priority-aware scheduling** for MoE-based LLM inference.
- It **preempts best-effort jobs** to prioritize **latency-sensitive requests**, significantly reducing TTFT and turnaround time.
- Demonstrates up to **101.6x lower TTFT** and **12.8x reduction in LS turnaround time**.
- Maintains **high throughput** while ensuring **SLO compliance**.
- Offers a **modular, extensible framework**, easily integrable with existing Hugging Face MoE models.

Thank You