# *throttLL'eM*
# Predictive GPU Throttling for Energy Efficient LLM Inference Serving

**Andreas K. Kakolyris** Dimosthenis Masouros
Petros Vavaroutsos Sotirios Xydis Dimitrios Soudris

ETH zürich

# Executive Summary

**Problem:** LLM inference consumes significant energy.

- Energy consumption is predicted to **increase** with further adoption
- Static optimization policies **violate** Service Level Objectives (SLOs)

**Goal:** Reduce the **energy consumption** of LLM inference serving **without violating SLOs**

**Key Idea:** **Predict** the future state of the serving system to find the **minimum performance level** required to achieve SLOs.

**Key Mechanism:** ***throttLL'eM***

- Models the token generation latency based on system metrics.
- Predicts how these system metrics will evolve over time.
- Adjusts the parameters of the system to minimize energy consumption while meeting SLOs.

**Key Result:** ***throttLL'eM*** can reduce the energy consumption of LLM serving by **42%.**

# Outline

Background

Motivation

*throttLL'eM*: Mechanism

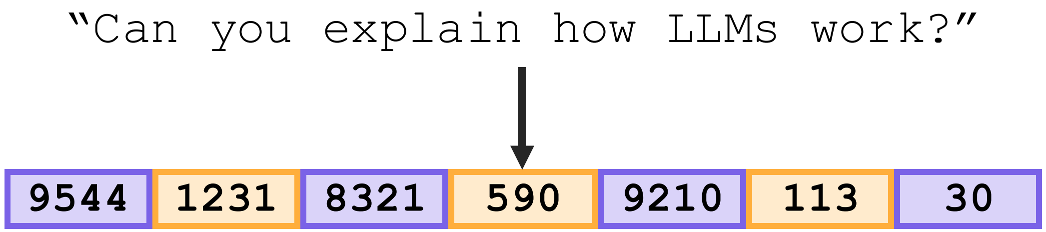
Evaluation

Conclusion

# Outline

- **Background**
- Motivation
- *throttLL'eM*: Mechanism
- Evaluation
- Conclusion

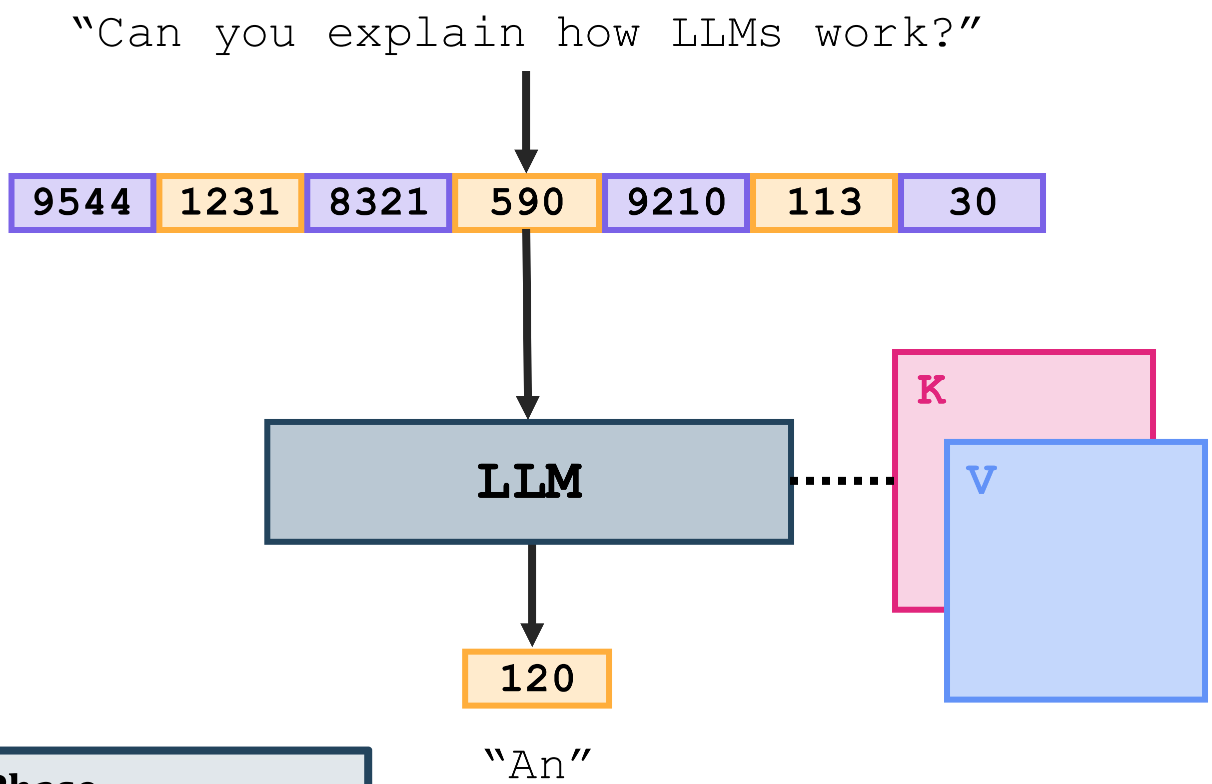
# Background on LLM inference




**1. Prompt Tokenization**

- Convert input (sub-)words to a unique representation (tokens)

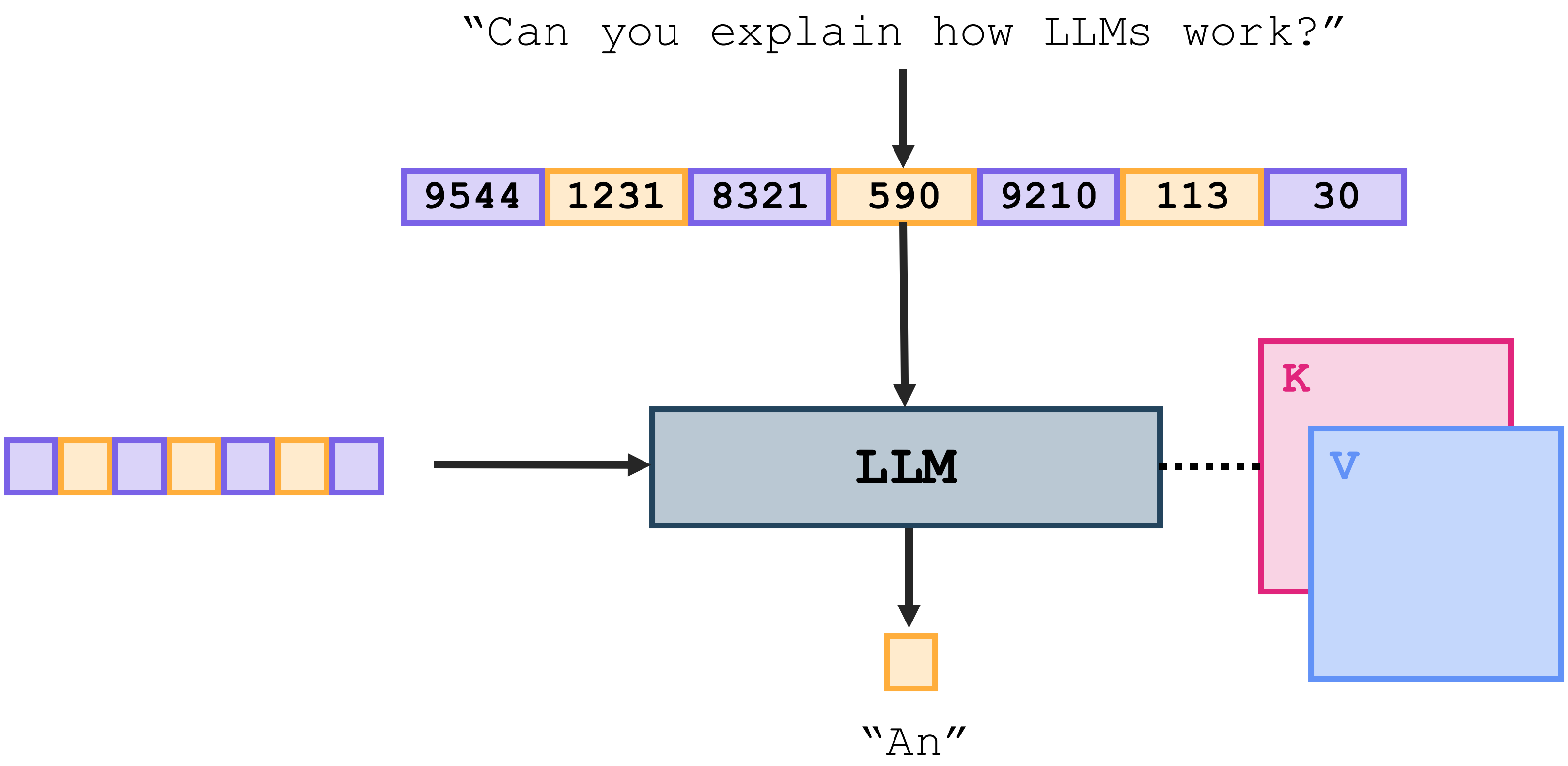
# Background on LLM inference



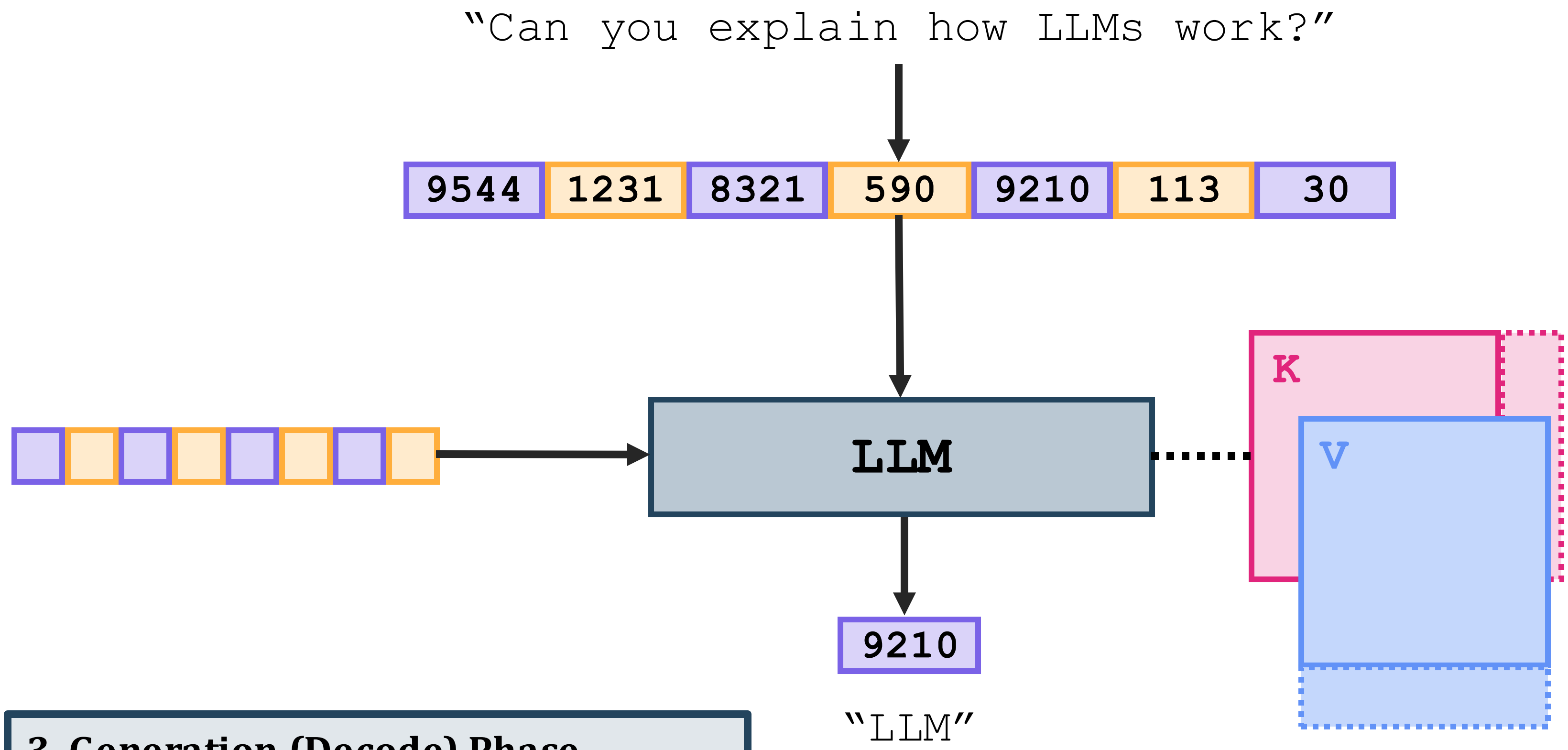

**2. Prefill (Prompt) Phase**

- Generate the first token of the answer
- Generate KV cache
- Compute bound

# Background on LLM inference



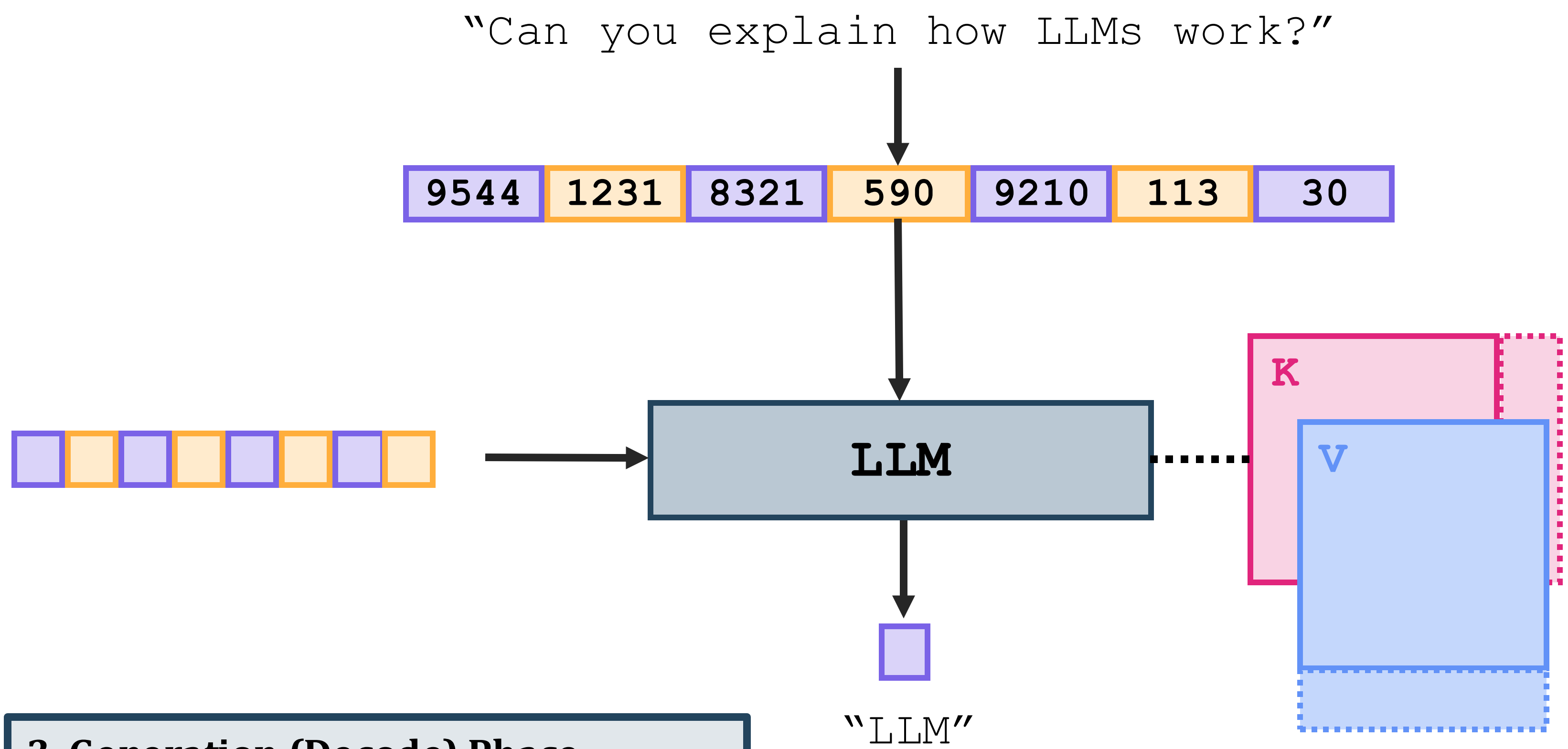
# Background on LLM inference




**3. Generation (Decode) Phase**

- Generate rest of response tokens
- Update KV cache
- Memory bound

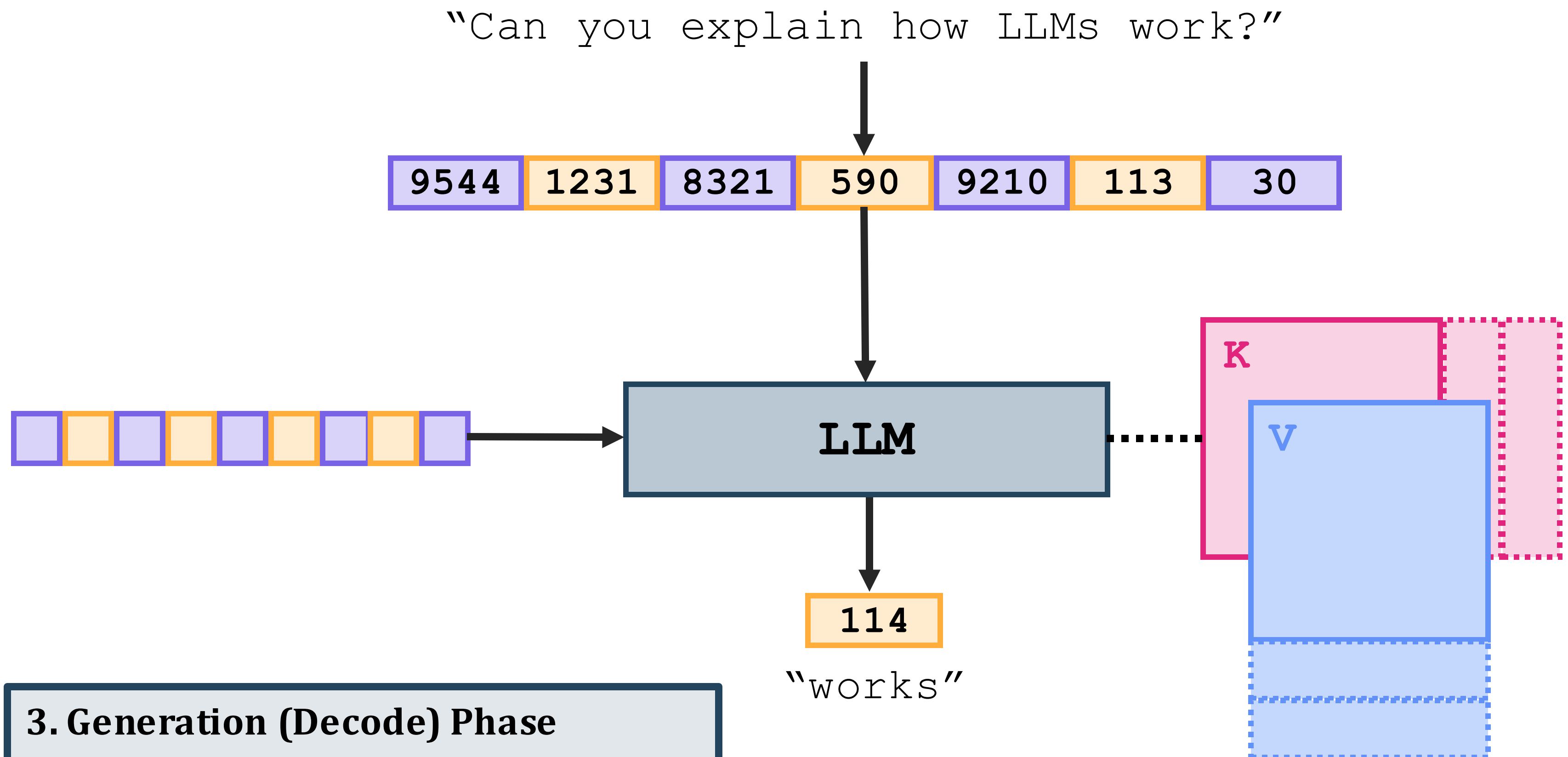
# Background on LLM inference




**3. Generation (Decode) Phase**

- Generate rest of response tokens
- Update KV cache
- Memory bound

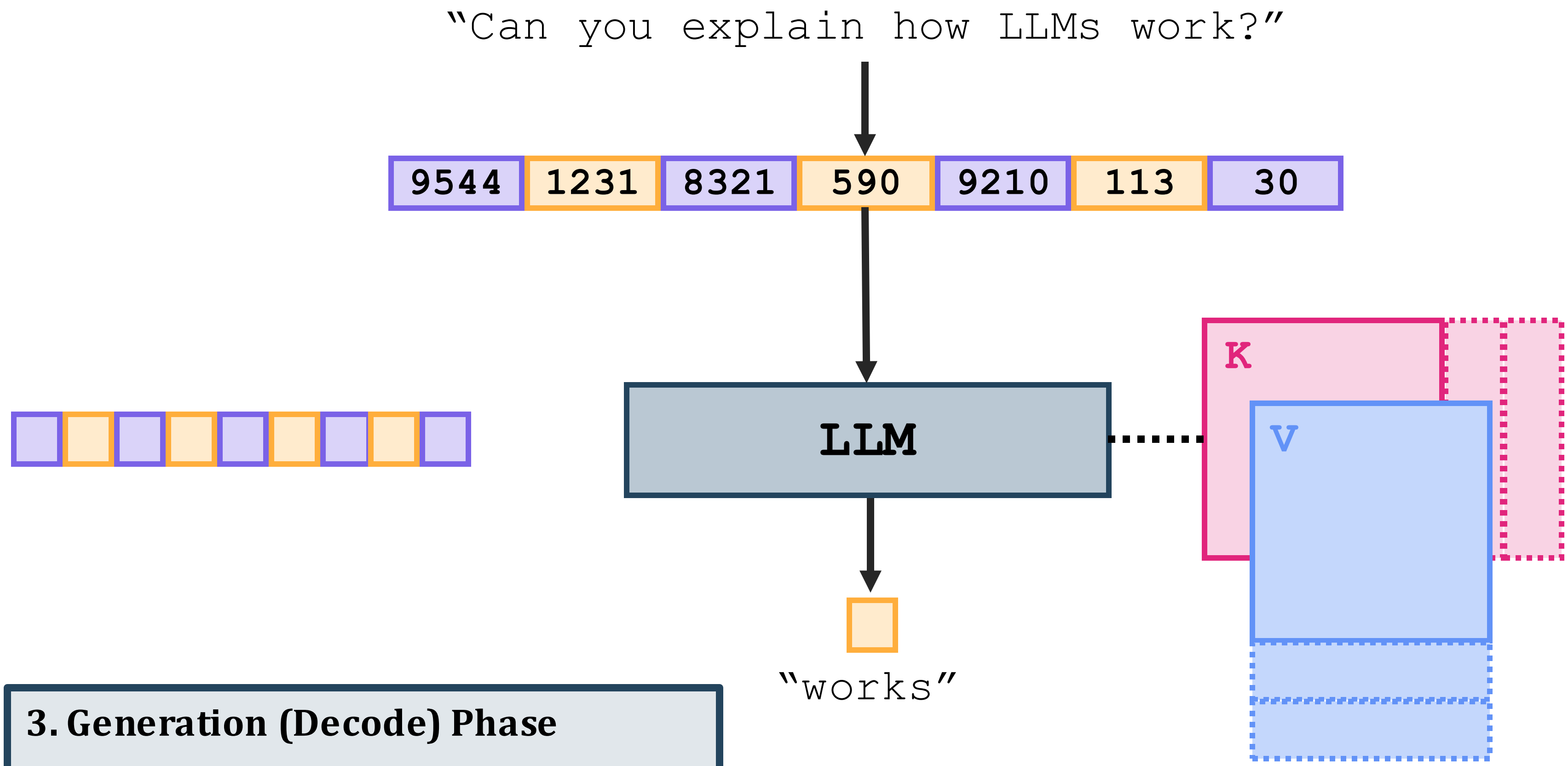
# Background on LLM inference




**3. Generation (Decode) Phase**

- Generate rest of response tokens
- Update KV cache
- Memory bound

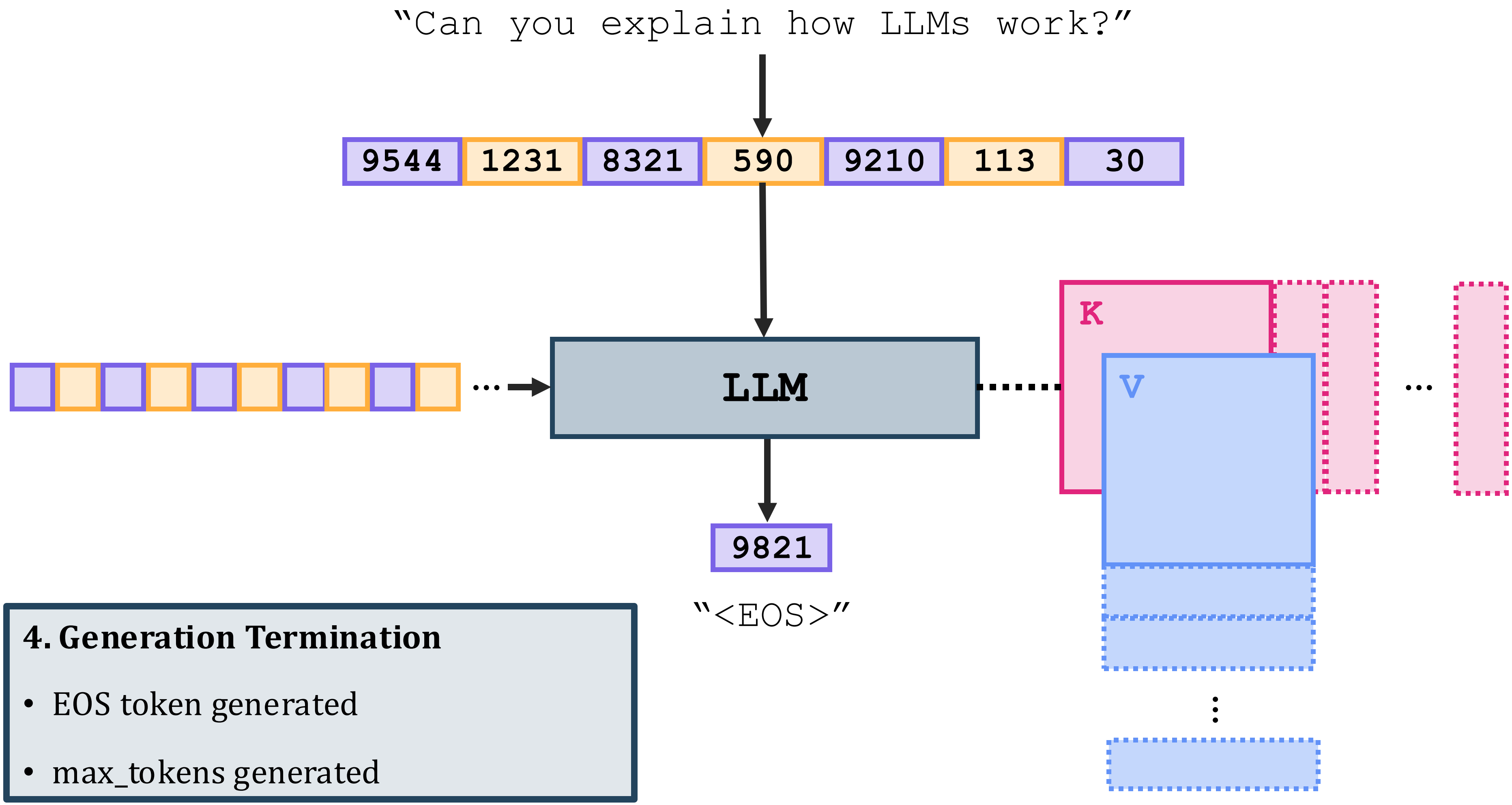
# Background on LLM inference




**3. Generation (Decode) Phase**

- Generate rest of response tokens
- Update KV cache
- Memory bound

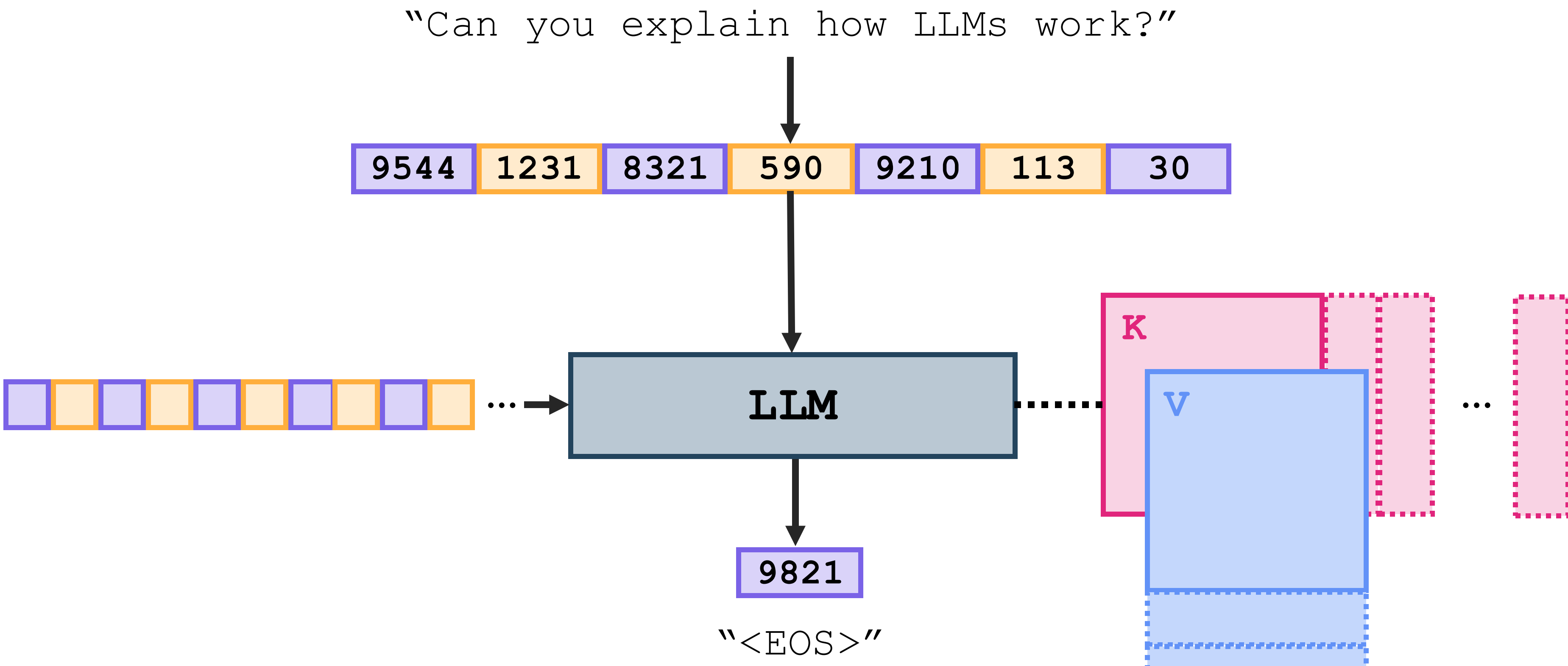
# Background on LLM inference




**4. Generation Termination**

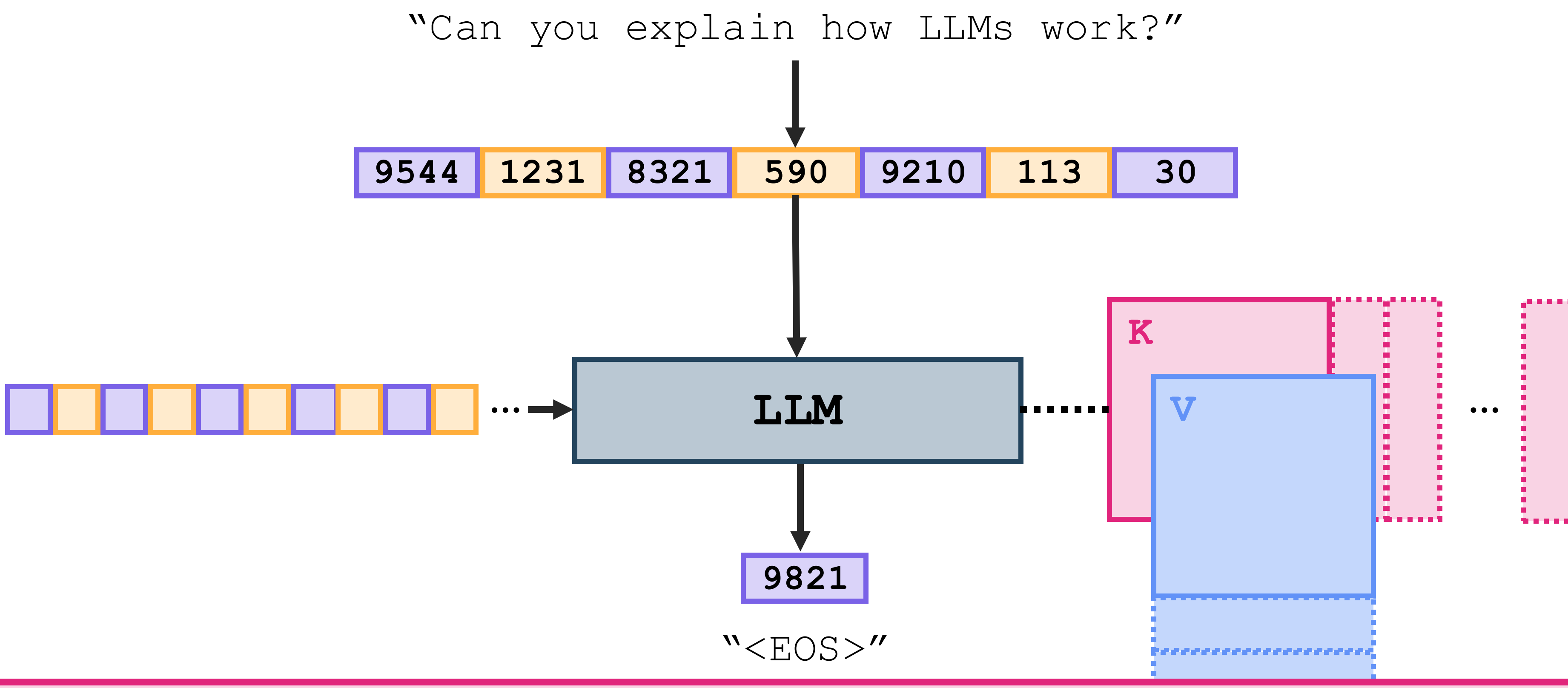
- EOS token generated
- max_tokens generated

# Challenges of LLM inference




**Autoregressive Token Generation ➔ Unpredictability**

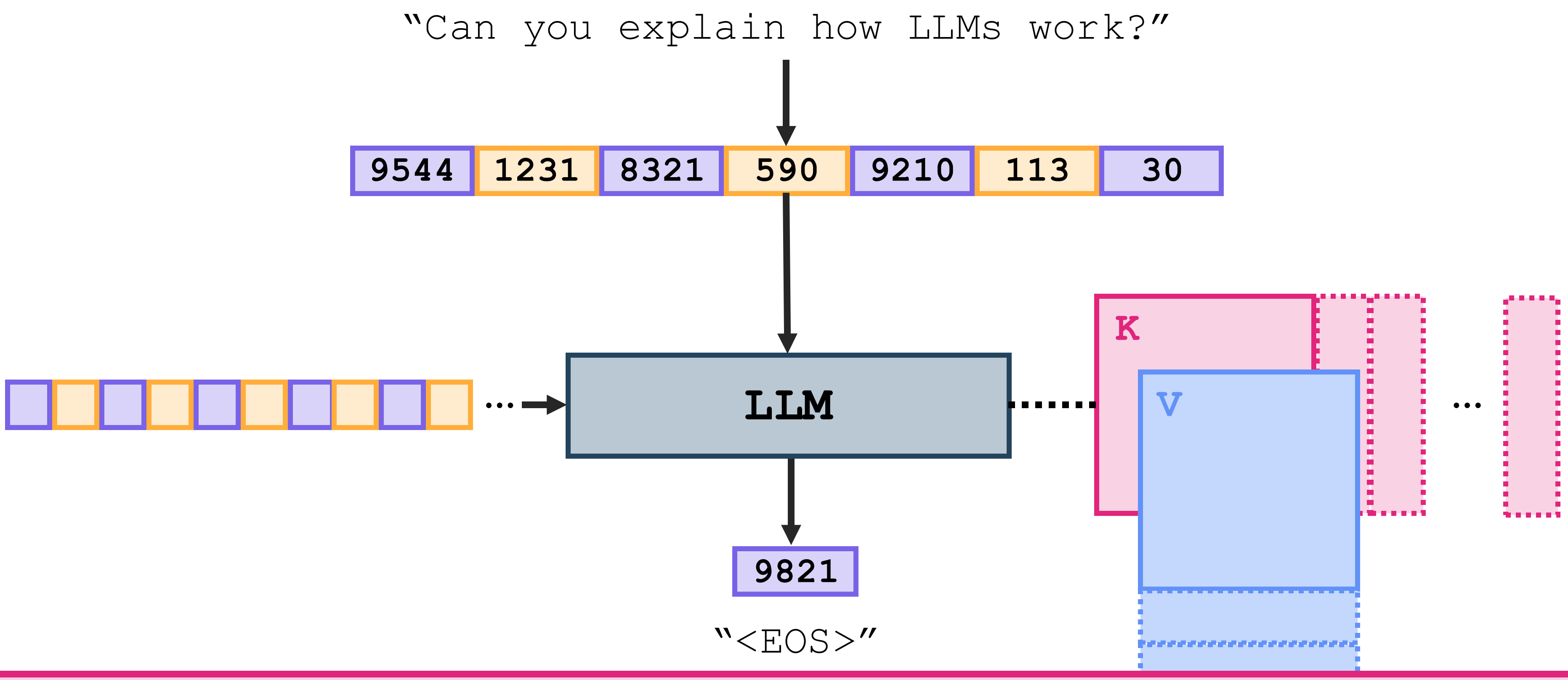
# Challenges of LLM inference




**Autoregressive Token Generation ➔ Unpredictability**

**Variable Memory Footprint ➔ Performance Variability**

# Challenges of LLM inference




**Autoregressive Token Generation ➔ Unpredictability**

**Variable Memory Footprint ➔ Performance Variability**

**Inflight Batching [Yu+, OSDI'22] ➔ Additional Performance Variability**

# Outline

Background

Motivation

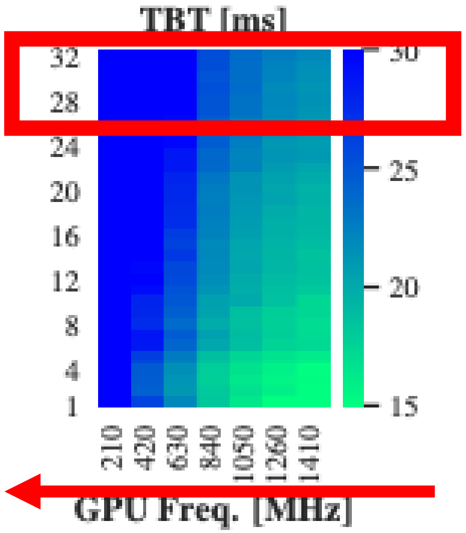
*throttLL'eM*: Mechanism

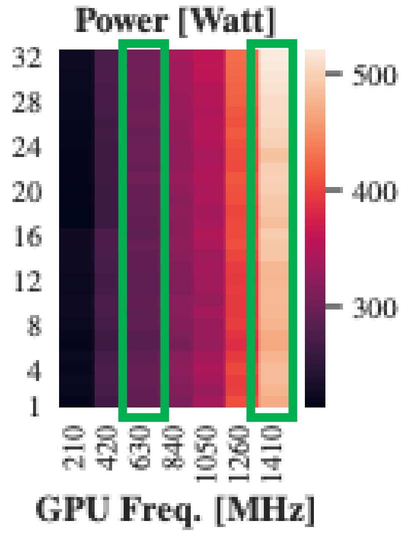
Evaluation

Conclusion

# Motivation:
# Performance-Energy Tradeoffs in LLM inference

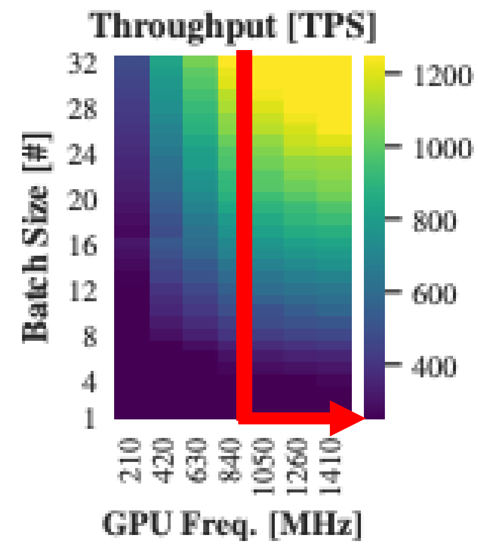
- Lower frequencies increase the Time-Between-Tokens
- Performance degradation decreases when using larger batch sizes
- GPU power draw only depends on the frequency used

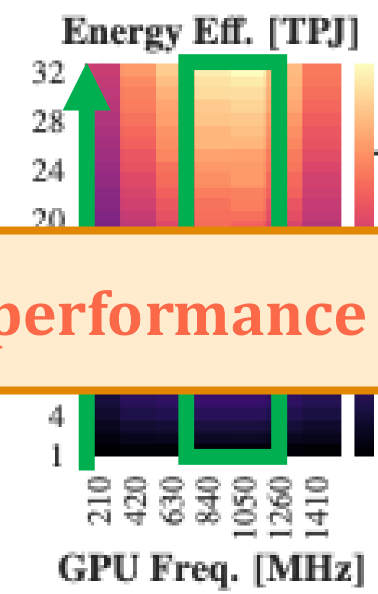


# Motivation:
# System Level Performance-Efficiency Tradeoffs

- Throughput depends on Batch size
- Performance gains diminish when using increasingly higher frequencies
- Energy Efficiency also increases with batch size
- Highest energy efficiency in the

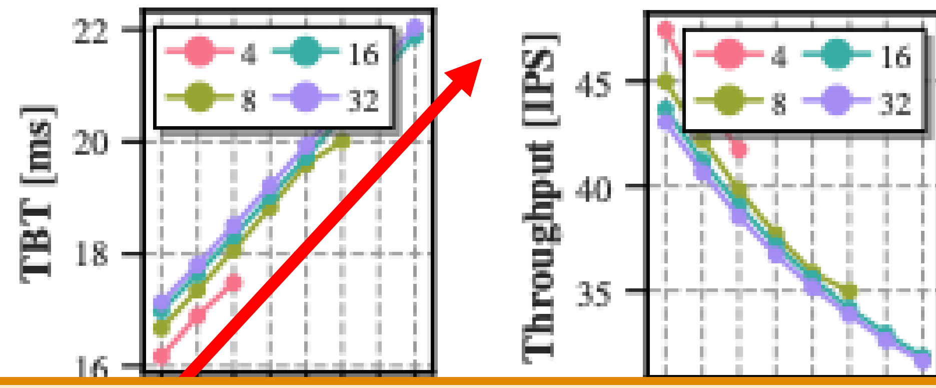



**High efficiency is possible with minimal performance loss**

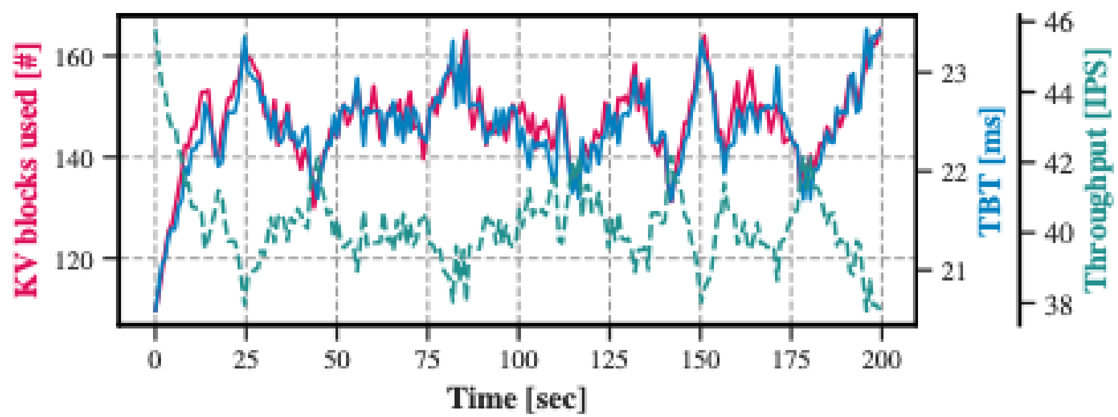
# Motivation: Modeling LLM performance

Inference slows down as context length increases




**KV cache size is an accurate proxy for performance**

- Constant Batch size
- Pearson Correlation of 0.92



# Motivation: Energy Efficient LLM serving

**Goal**

**Reduce energy consumption while meetings SLOs**

**Idea**

**Model performance at the iteration level to enable fine-grained energy efficiency optimization**

# Outline

Background

Motivation

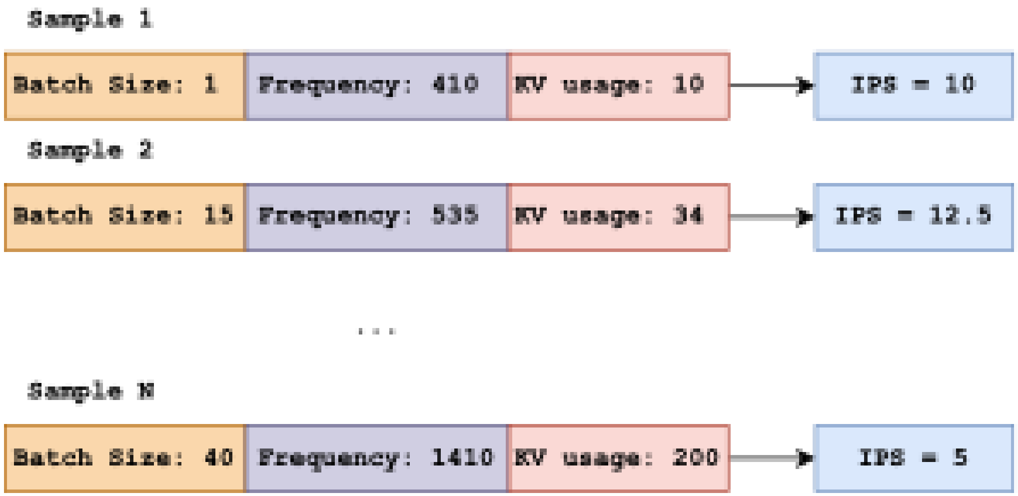
*throttLL'eM*: Mechanism

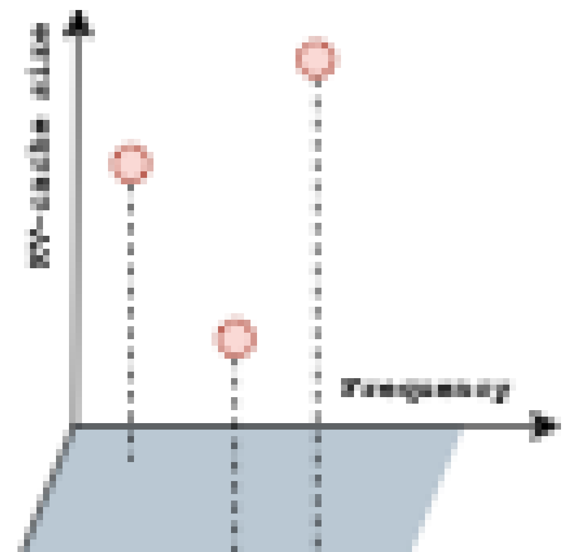
Evaluation

Conclusion

# *throttLL'eM*
# Modelling System Performance

***throttLL'eM*** **sweeps** batch size, **logs** KV cache size and performance using **randomly** chosen frequencies

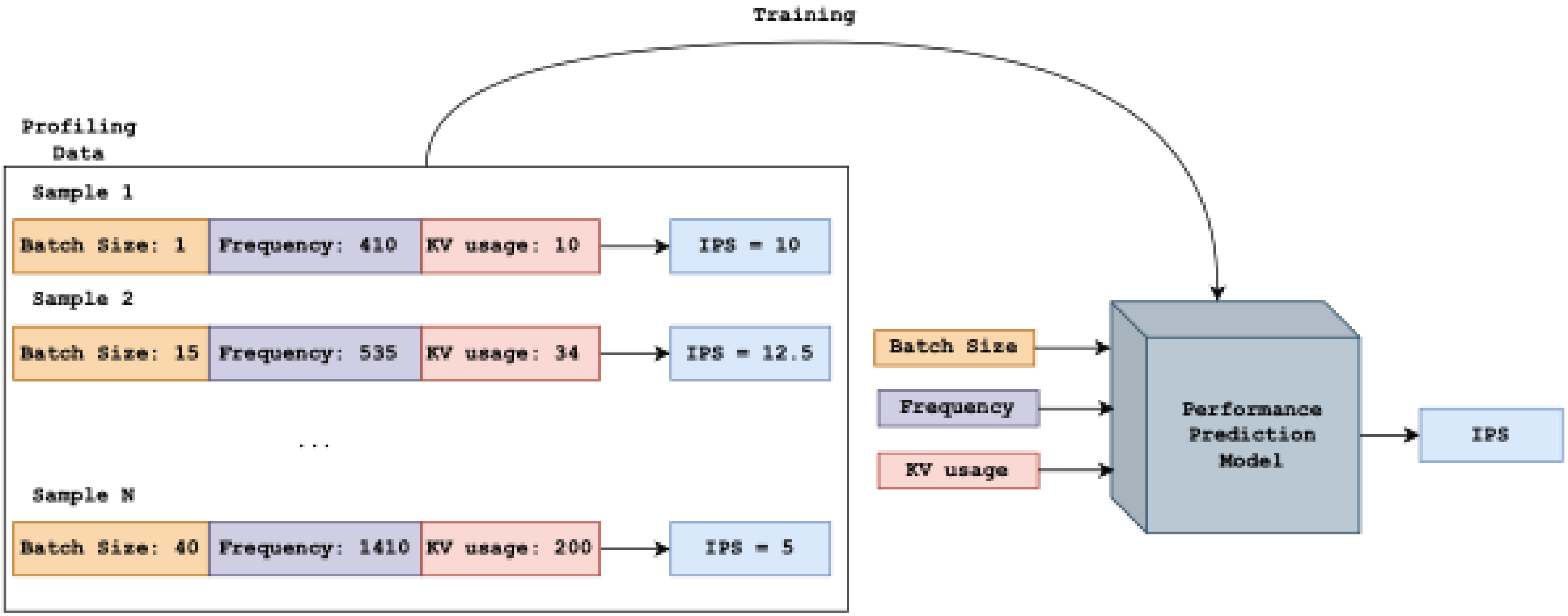



**The gathered samples cover the entire system state space**

# *throttLL'eM*
# Modelling System Performance

***throttLL'eM*** trains a **Machine Learning** model that predicts performance using the gathered samples



# *throttLL'eM*: Online Stage

1) Predicting future states

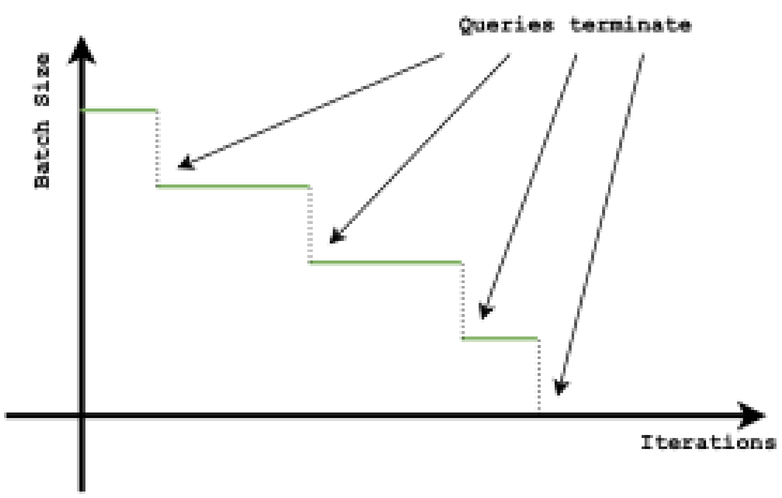
2) Validate SLOs

3) Adjust System Performance

# *throttLL'eM*
# Predicting future states

***throttLL'eM*** employs a generation length prediction model to predict how many tokens a query will generate

***throttLL'eM*** uses these predictions to forecast batch size and KV cache size at each future iteration



# *throttLL'eM*: Online Stage

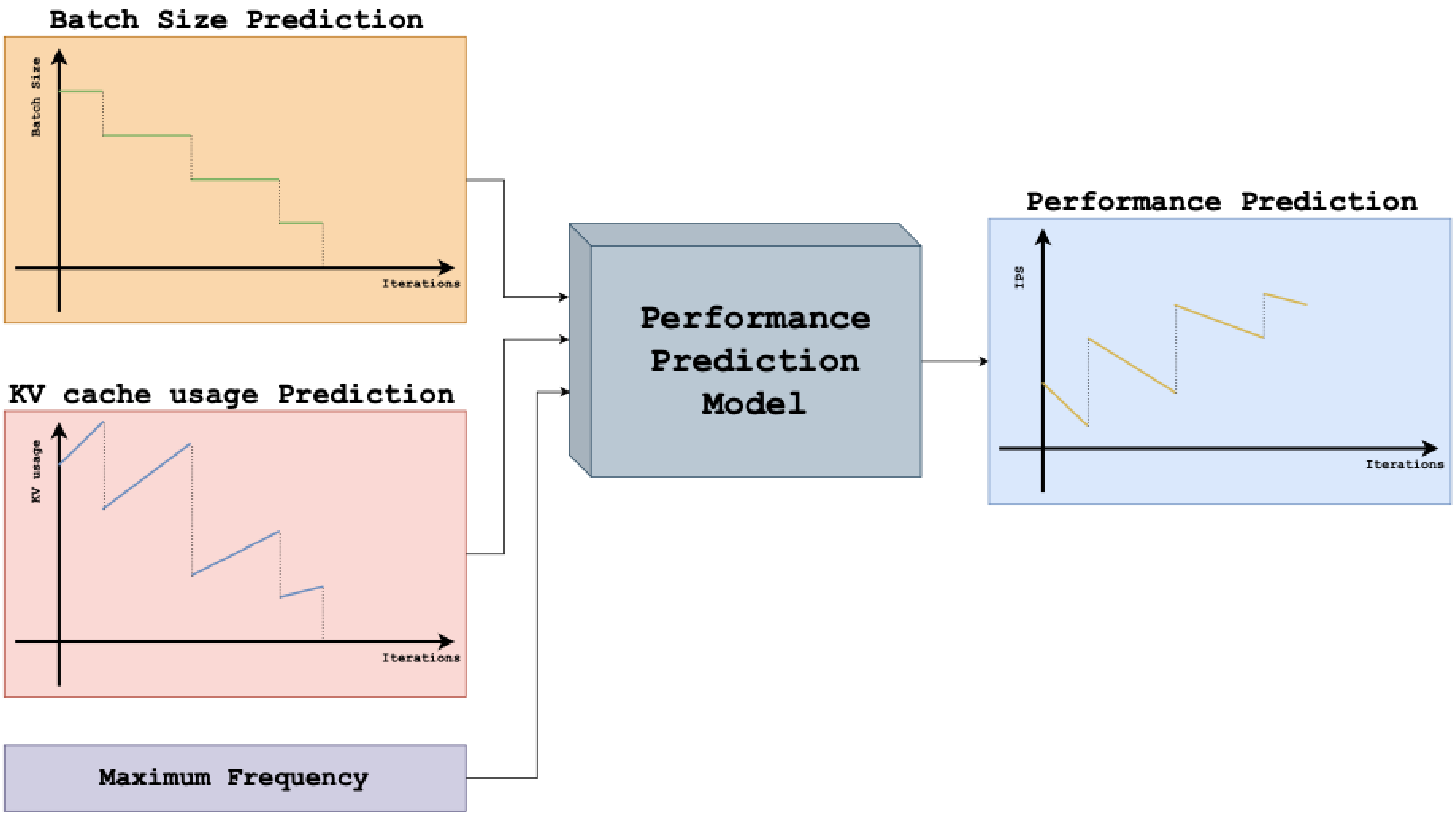
1) Predicting future states

2) Validate SLOs

3) Adjust System Performance

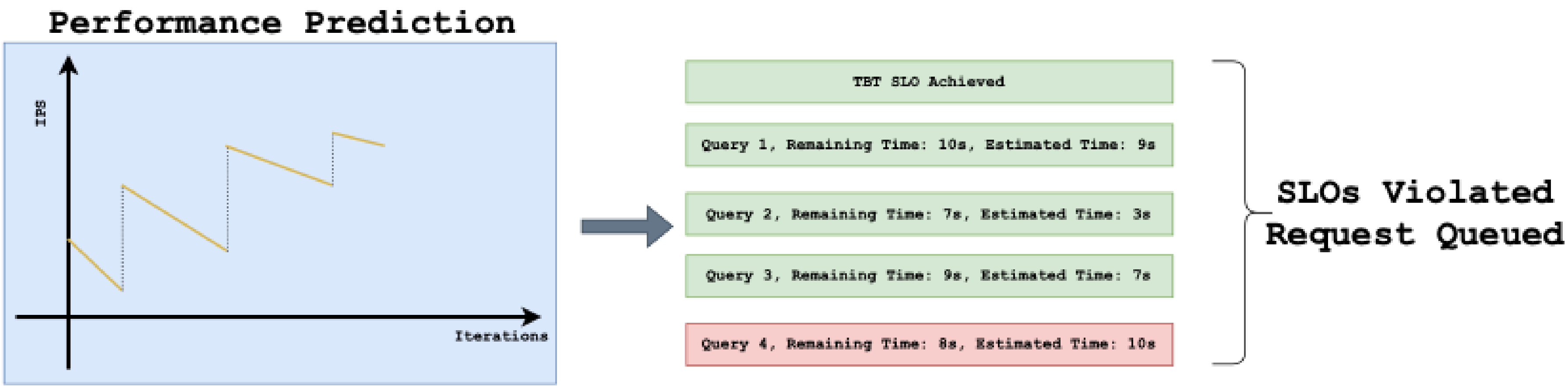
# *throttLL'eM*
# Validating SLOs

Before scheduling a request, ***throttLL'eM*** predicts its impact on the future performance of the system



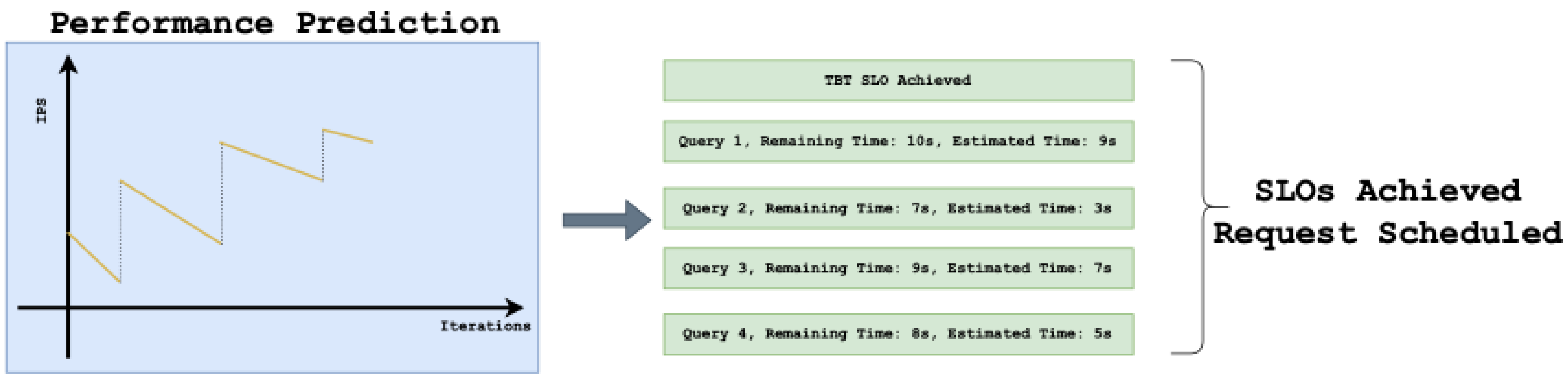
# *throttLL'eM*
# Validating SLOs

***throttLL'eM*** uses the performance predictions to check if the SLOs of running requests can be attained if the request is scheduled



# *throttLL'eM*
# Validating SLOs

***throttLL'eM*** uses the performance predictions to check if the SLOs of running requests can be attained if the request is scheduled



# *throttLL'eM:* Online Stage

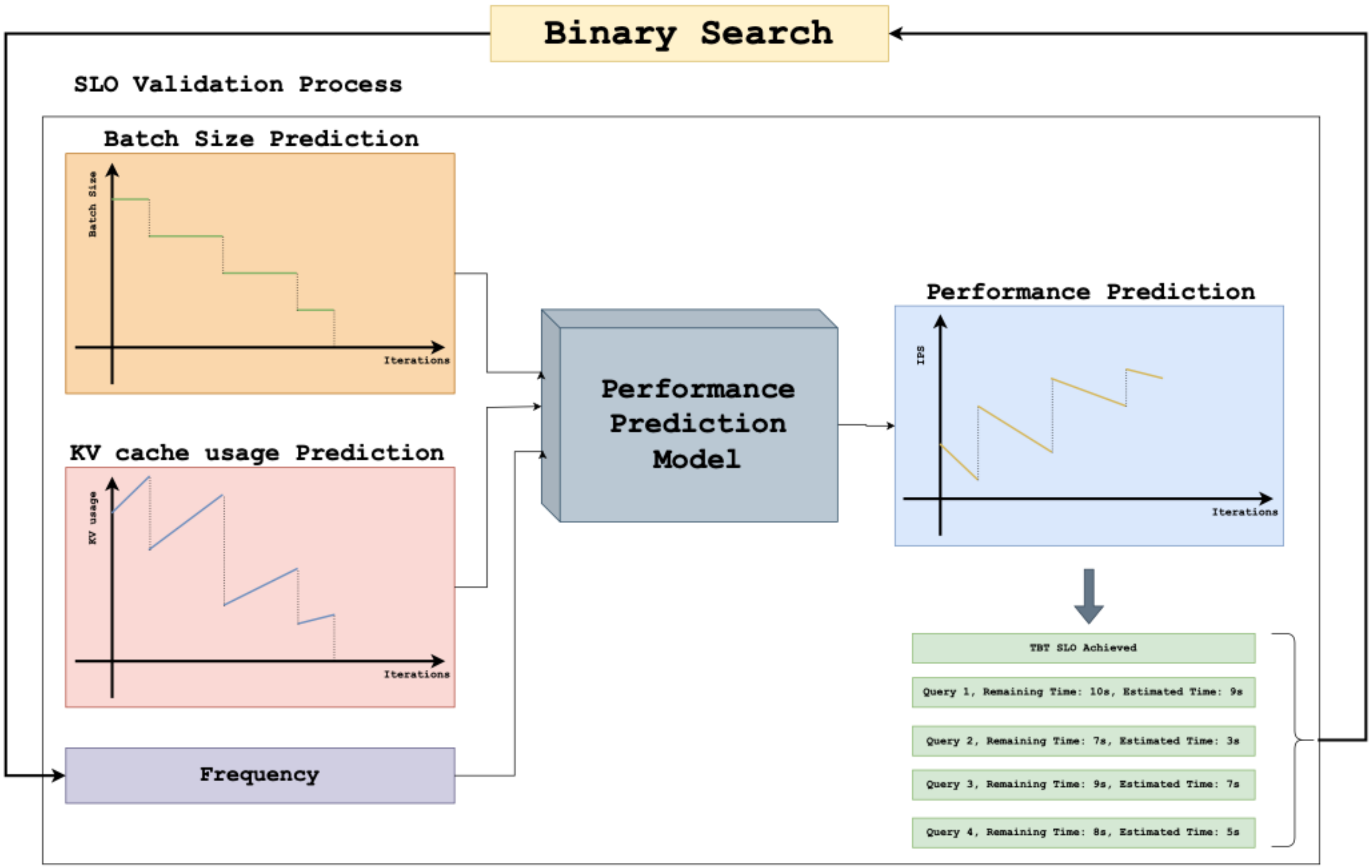
1) Predicting future states

2) Validate SLOs

3) Adjust System Performance

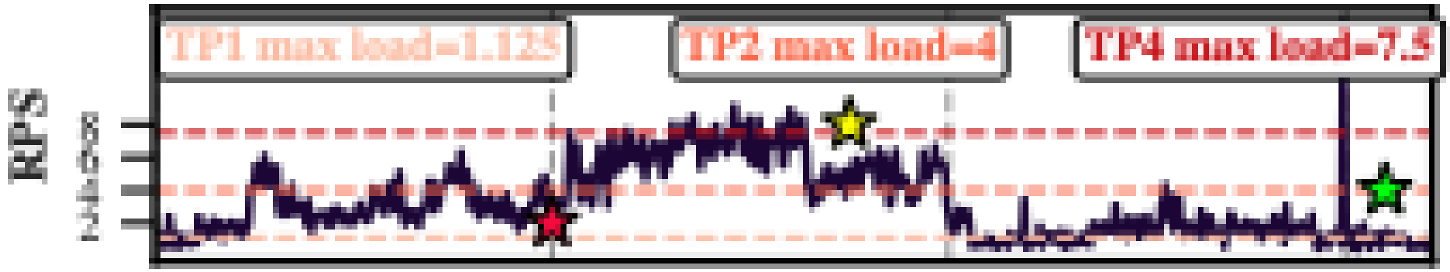
# *throttLL'eM*
# Adjusting System Performance

***throttLL'eM*** performs a binary search of the Frequency search space to find the minimum frequency that satisfies **all SLOs**

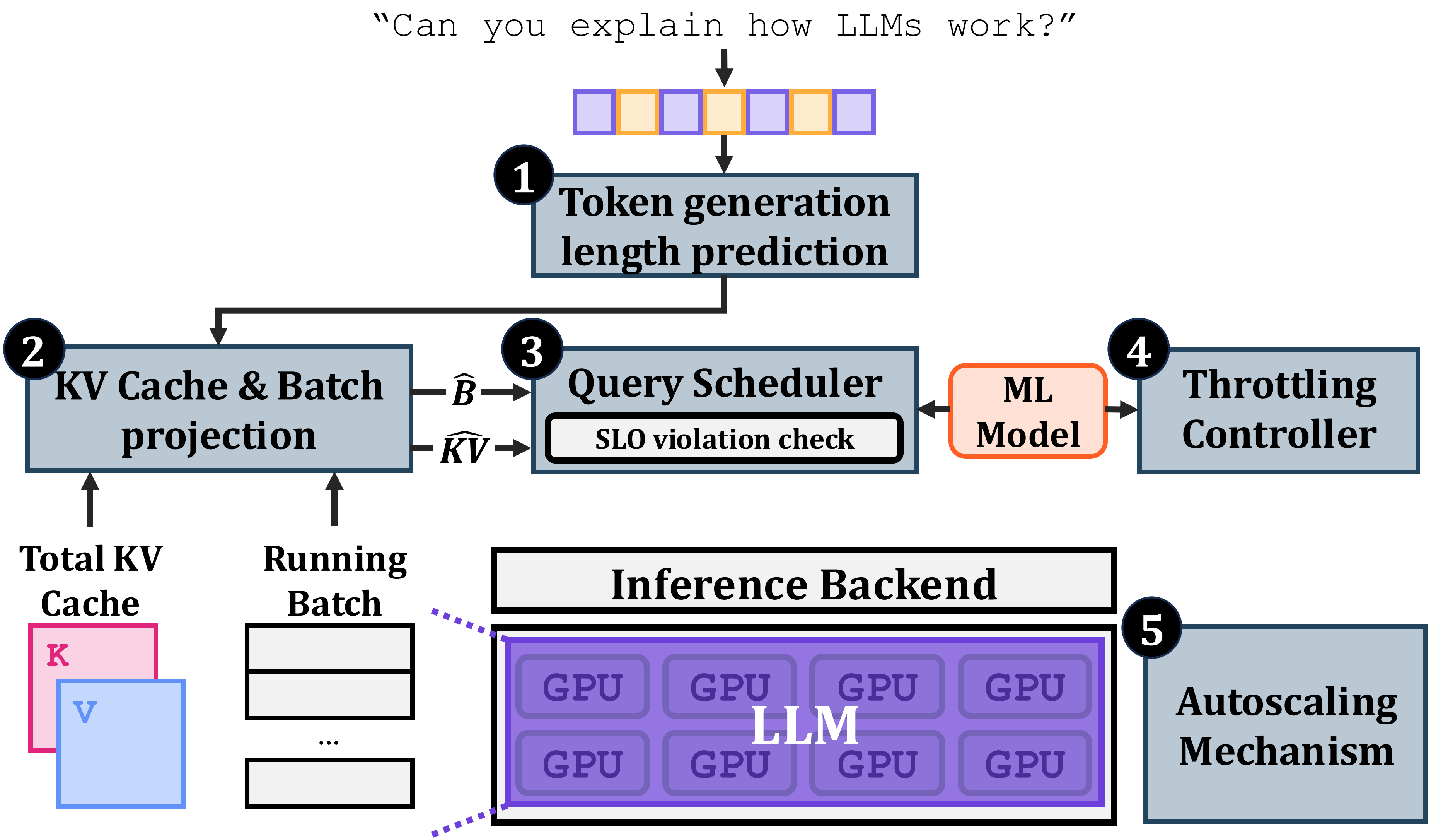


# *throttLL'eM*
# Adjusting System Performance

***throttLL'eM*** periodically checks and scales the capacity of the system using predetermined **load thresholds**



# *throttLL'eM:* Overview



# Outline

Background

Motivation

*throttLL'eM*: Mechanism

Evaluation

Conclusion

# Evaluation Methodology

- **System Configuration: NVIDIA DGX-A100**

| | |
|---|---|
| **Processor** | **2x** AMD EPYC 7742 |
| **DRAM** | **1TB** DDR4 |
| **GPUs** | **8x** NVIDIA A100-SXM4-40GB |
| **Software** | **NVIDIA** Triton + TensorRT-LLM backend |

- **Evaluated LLMs: LLaMa family of models**

| | |
|---|---|
| **LLaMa3 8B** | TP1 configuration |
| **LLaMa2 13B** | TP1, TP2 and TP4 configurations |
| **LLaMa3 70B** | TP8 configuration |

- **LLM Inference Trace:**
  - Inference trace from Azure
  - Contains query input and generation lengths
  - Time-scaled to match the peak performance of the evaluated system

# Evaluation Results

1) Performance Modeling Evaluation

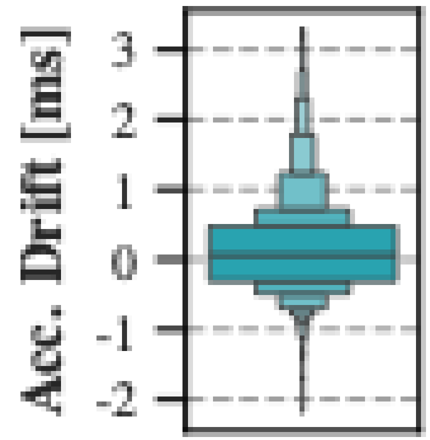
2) Frequency Scaling Evaluation

3) End-to-End throttLL'eM evaluation

# Performance Modeling Evaluation

| | *train = 90%* | | | *train = 10%* | | |
|---|---|---|---|---|---|---|
| | **$R^2$** (-) | **MAPE** (%) | **MAE** (iters/s) | **$R^2$** (-) | **MAPE** (%) | **MAE** (iters/s) |
| **Llama3-8B-TP1** | 0.99 | 4.1 | 0.85 | 0.98 | 4.2 | 0.93 |
| **Llama2-13B-TP1** | 0.98 | 2.8 | 0.74 | 0.97 | 3.0 | 0.79 |
| **Llama2-13B-TP2** | 0.99 | 3.1 | 0.90 | 0.99 | 3.4 | 0.99 |
| **Llama2-13B-TP4** | 0.99 | 3.3 | 0.97 | 0.99 | 3.4 | 1.01 |
| **Llama3-70B-TP8** | 0.97 | 5.8 | 0.69 | 0.96 | 6.5 | 0.77 |

$R^2$ score, MAPE and MAE for different train-test splits and model configurations




Distribution of accumulated drift per elapsed iteration

**The performance prediction model achieves high performance, even with sparse datasets**

***throttLL'eM*** **accumulates a relatively small average drift of 0.43ms per iteration**

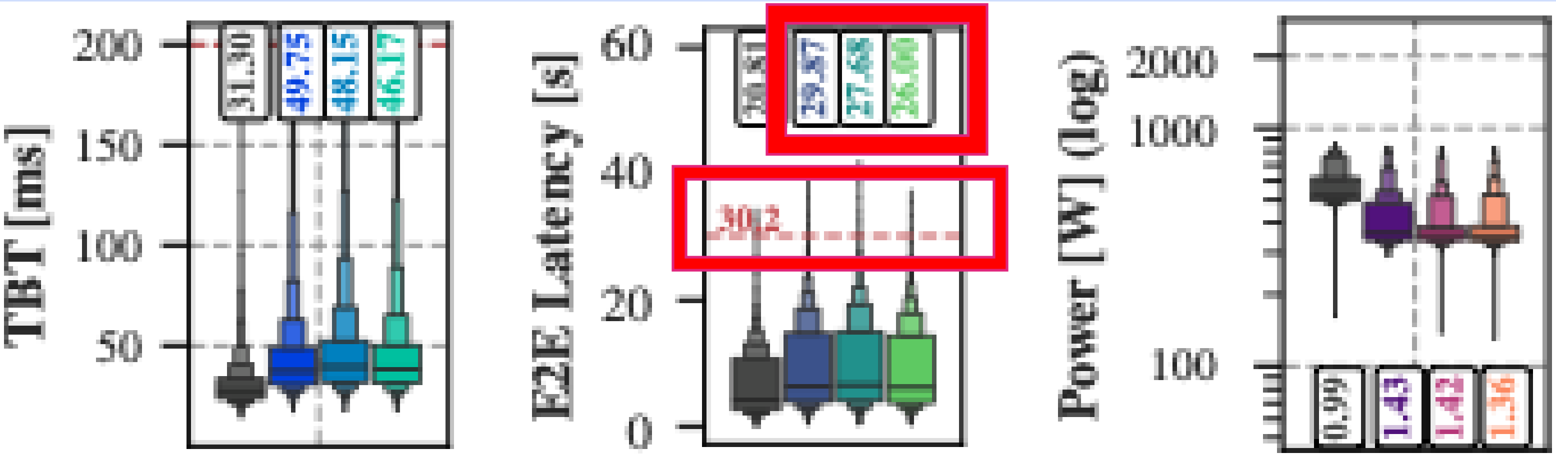
# Evaluation Results

1) Performance Modeling Evaluation

2) Frequency Scaling Evaluation

3) End-to-End *throttLL'eM* evaluation

# Frequency Scaling Evaluation

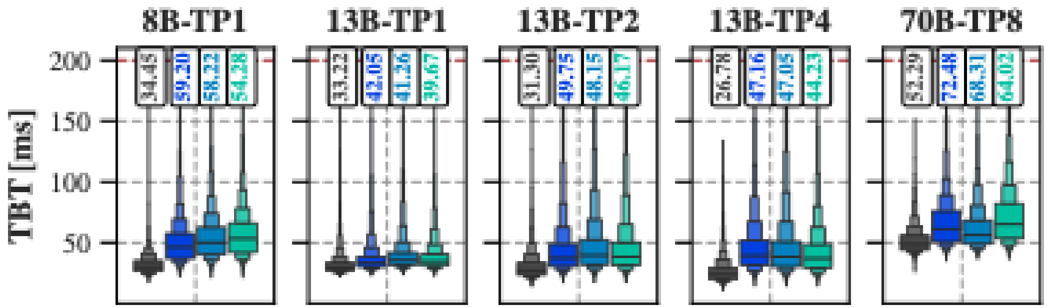



Distribution of **a)** Time-between-Tokens, **b)** End-to-End latency and **c)** Power consumption for the default implementation and throttLL'eM at 0%, 15% and 30% error levels for LLaMa2-13B-TP2

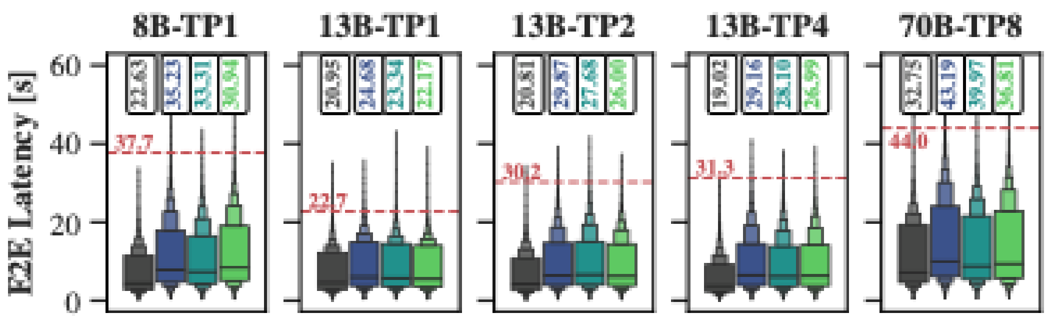
**As the error level increases, *throttLL'eM* becomes more conservative, leading to lower energy efficiency**

***throttLL'eM* significantly increases energy efficiency even at 30% prediction error level**

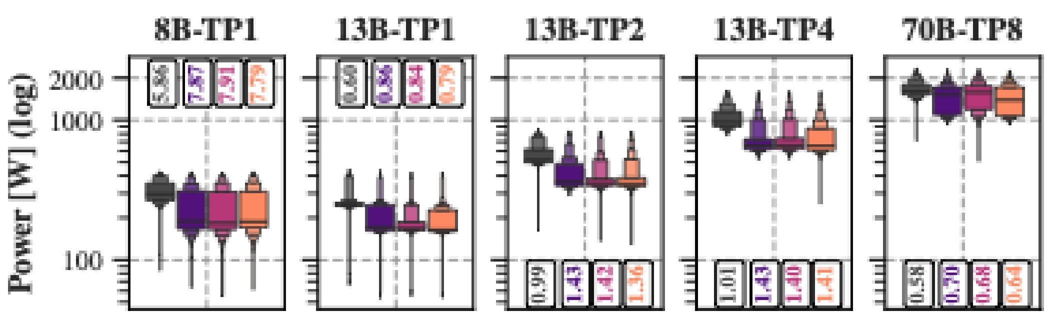
# Frequency Scaling Evaluation




Distribution of Time-between-Tokens for different models and configurations

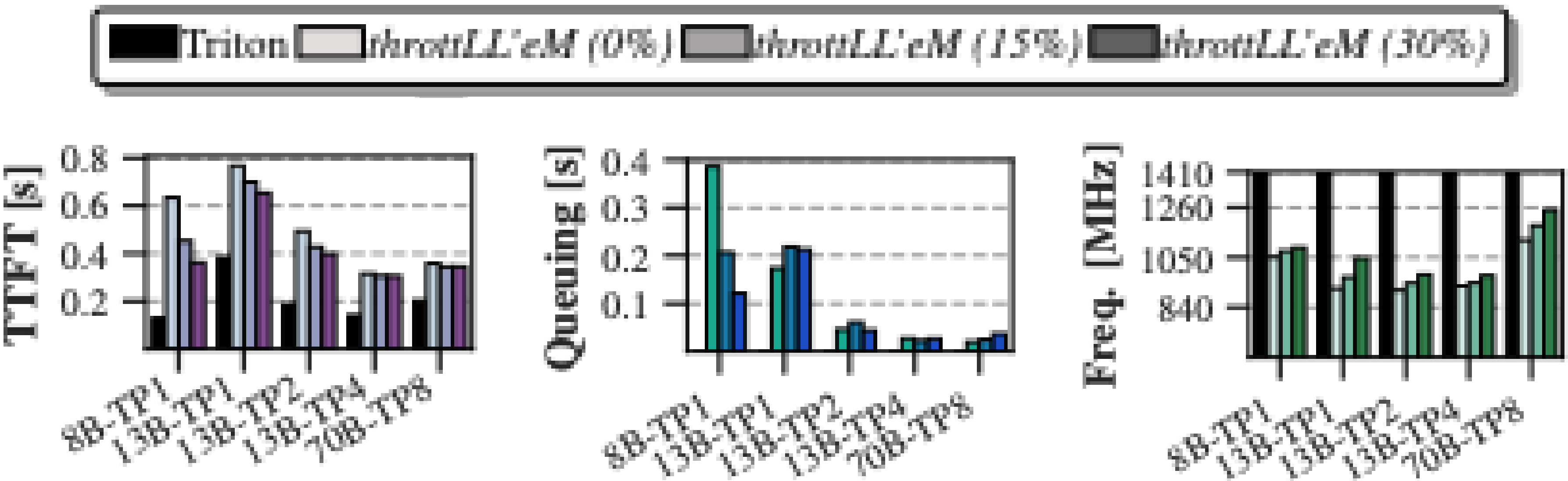



Distribution of End-to-End latency for different models and configurations




Distribution of Power draw for different models and configurations

# Frequency Scaling Evaluation




Time-to-First-Token for different models and configurations

Queueing time for different models and configurations

Average Frequency for different models and configurations

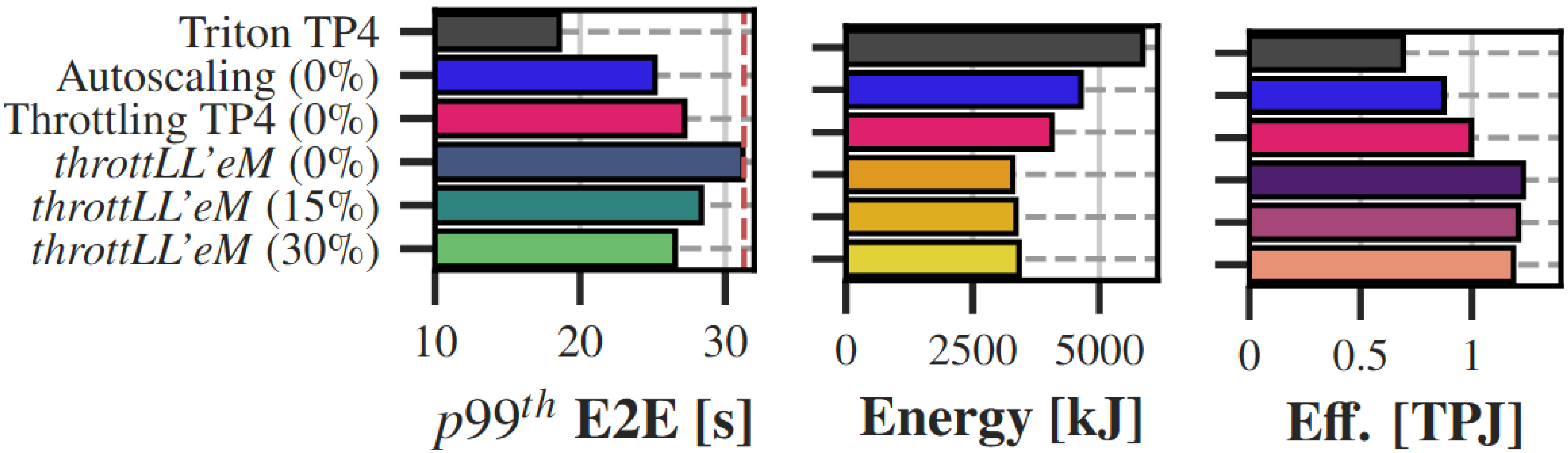
# Evaluation Results

1) Performance Modeling Evaluation

2) Frequency Scaling Evaluation

3) End-to-End *throttLL'eM* evaluation

# Ablation Study




***throttLL'eM* significantly increases energy efficiency by using both instance and frequency scaling**

Autoscaling → 20.8%

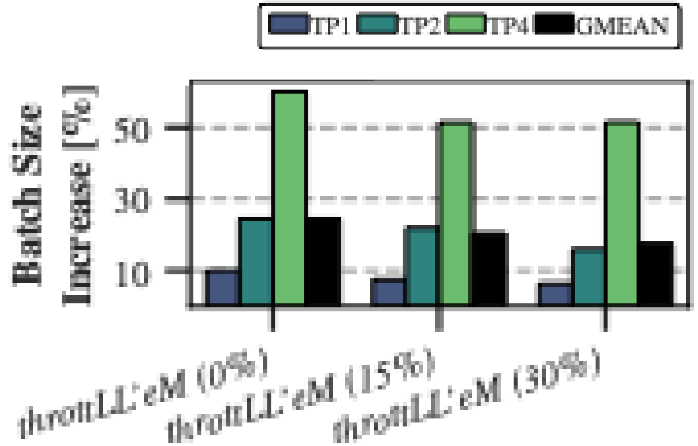
Frequency scaling → 30.6%

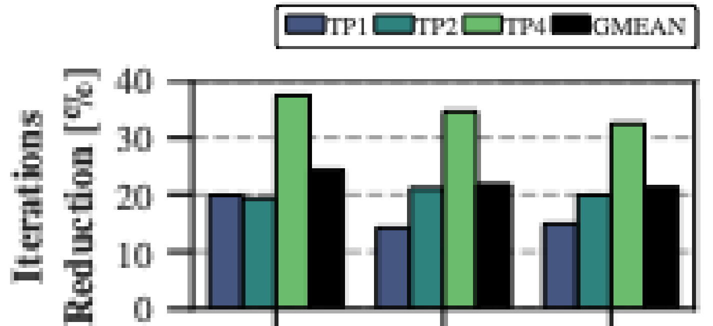
*throttLL'eM* → 41.7%.

# Result Interpretation

**By increasing the latency of individual LLM iterations:**

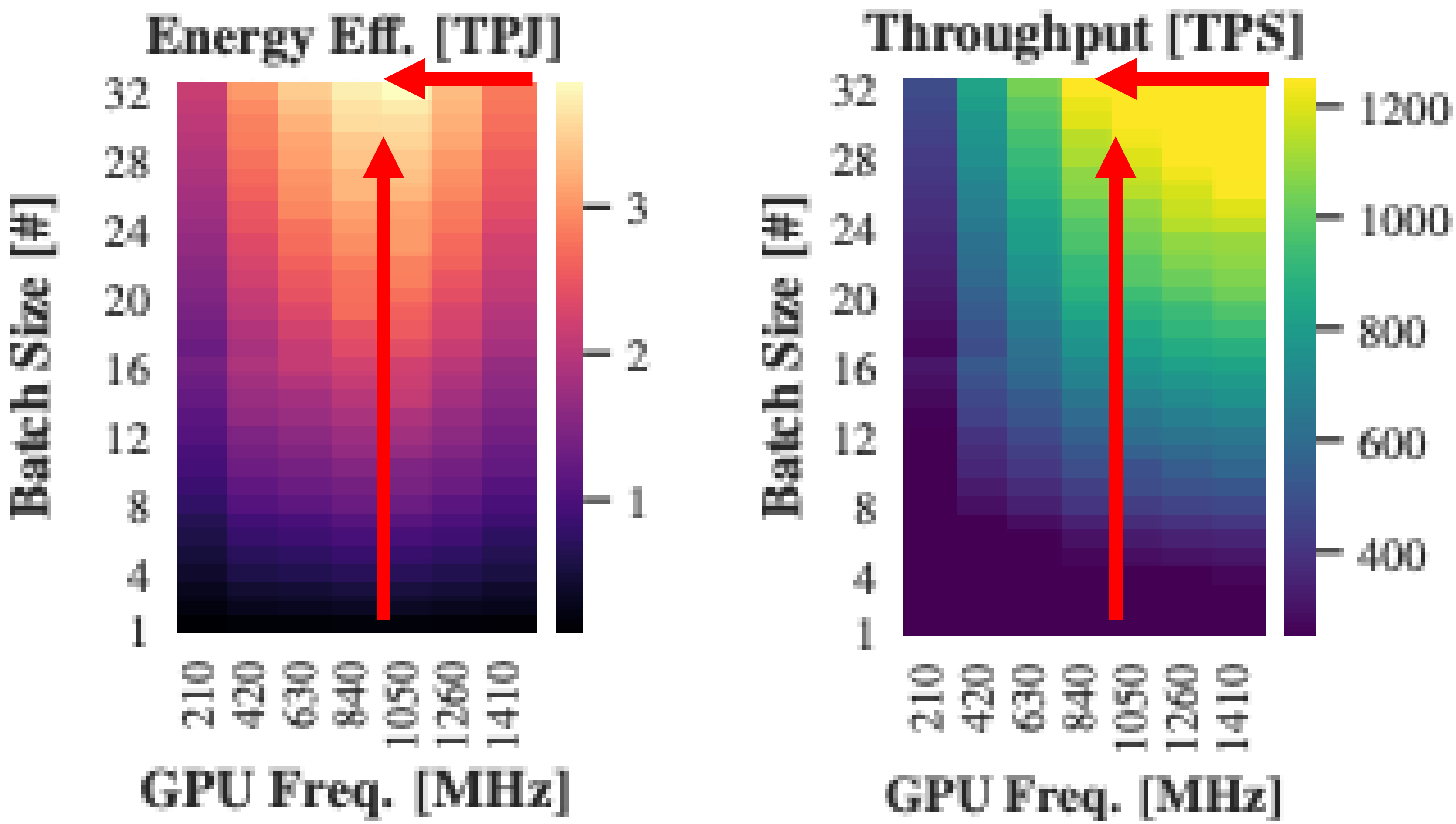
- Increases the average batch size
- Reduces the number of performed forward passes




***throttLL'eM*** **performs fewer LLM inferences by using a larger batch size, increasing efficiency**

# Motivation (again):
# System Level Performance-Efficiency Tradeoffs



# Outline

Background

Motivation

*throttLL'eM*: Mechanism

Evaluation

**Conclusion**

# Conclusion

*throttLL'eM*

- Accurately models LLM performance
- Predicts how the state of the system evolves over time
- Accordingly scales the frequency and the capacity of the system to reduce the energy consumption while meeting SLOs

**Key Results:**

- $R^2 > 0.97$
- Small per iteration performance modelling drift of 0.43ms
- Energy efficiency savings of upwards of 41%

# *throttLL'eM*
# Predictive GPU Throttling for Energy Efficient LLM Inference Serving

**Andreas K. Kakolyris** Dimosthenis Masouros
Petros Vavaroutsos Sotirios Xydis Dimitrios Soudris

ETH zürich

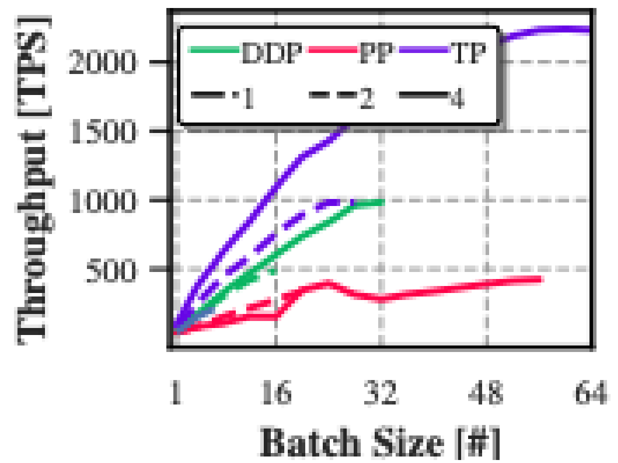
# *throttLL'eM*
# Predictive GPU Throttling for Energy Efficient LLM Inference Serving

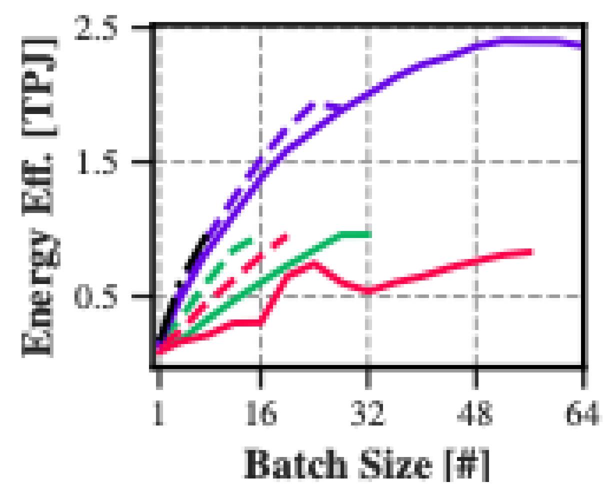
**Backup Slides**

**Andreas K. Kakolyris** Dimosthenis Masouros
Petros Vavaroutsos Sotirios Xydis Dimitrios Soudris

# Motivation:
# Modeling LLM performance

- Tensor Parallelism exhibits the highest throughput
- Tensor Parallelism exhibits the highest energy efficiency
- Minimizing the number of GPUs used is necessary for optimal energy efficiency

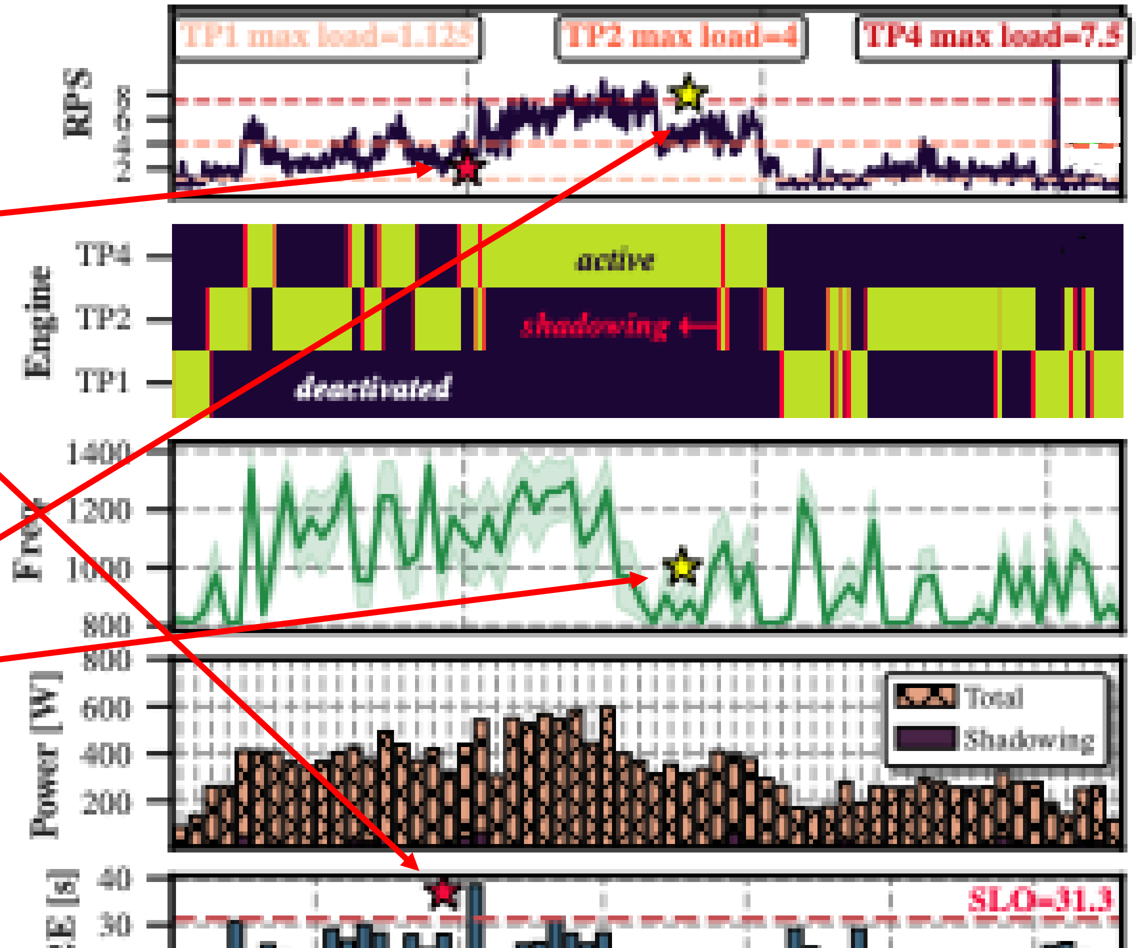


# Analysis on the trace

Transient E2E violations due to load increment while shadow instancing

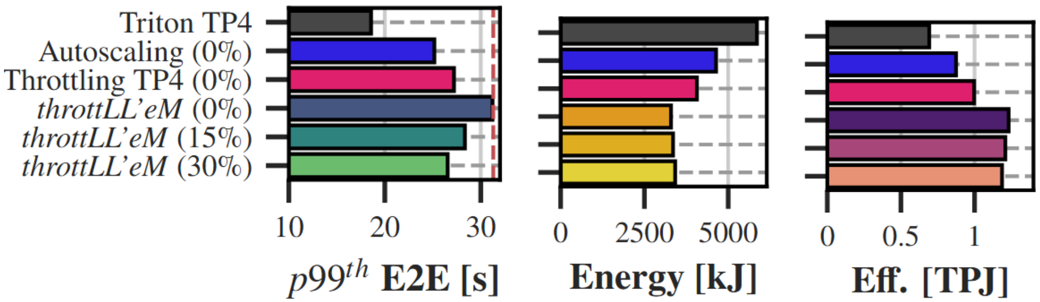
Aggressive Frequency reduction when an instance cannot be downscaled




**Autoscaling provides coarse-grained throughput adjustments**
**Frequency scaling provides finer control**

# Ablation study and Comparison

- **Autoscaling** reduces energy consumption by **20.8%**
- **Frequency scaling** reduces energy consumption by **30.6%**
- ***throttLL'eM*** reduces energy consumption by **41.7%** over the baseline.




- Compared against a Retail-like DVFS inspired implementation, throttLL'eM achieves XXX lower power consumption on average and approach the SLO deadline more aggressively.