A Hacker's Guide to Speculative Decoding in vLLM

CUDA MODE talk by Cade Daniel

Introductions

- Working on LLM inference in vLLM
- Software Engineer at Anyscale
- Previously, model parallelism systems at AWS
  - https://arxiv.org/abs/2111.05972
- Feel free to reach out!
  - https://x.com/cdnamz

Topics

- vLLM’s core principles
- Spec decode background
- Spec decode framework intro (for contributors)
- Future contribution ideas
- Q&A (30min)

vLLM’s core principles

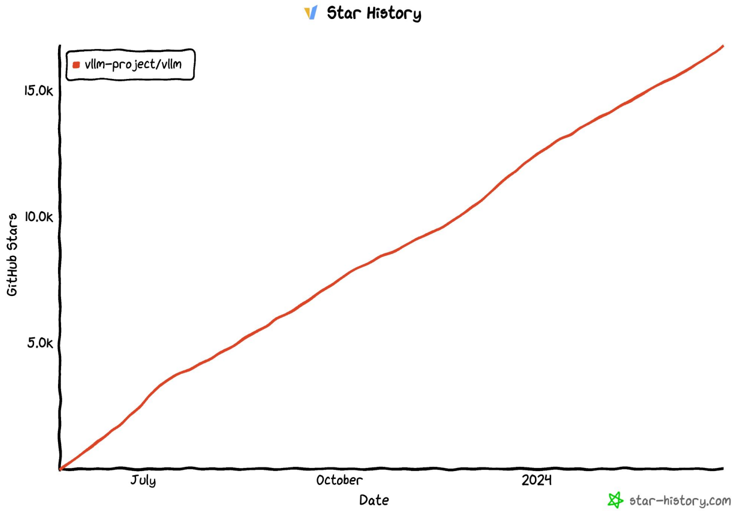
- Ease-of-use
- Great performance
- Hardware agnosticity

Easy-to-use

https://github.com/vllm-project/vllm

$ pip install vllm

17K Stars



Great performance

Performance features

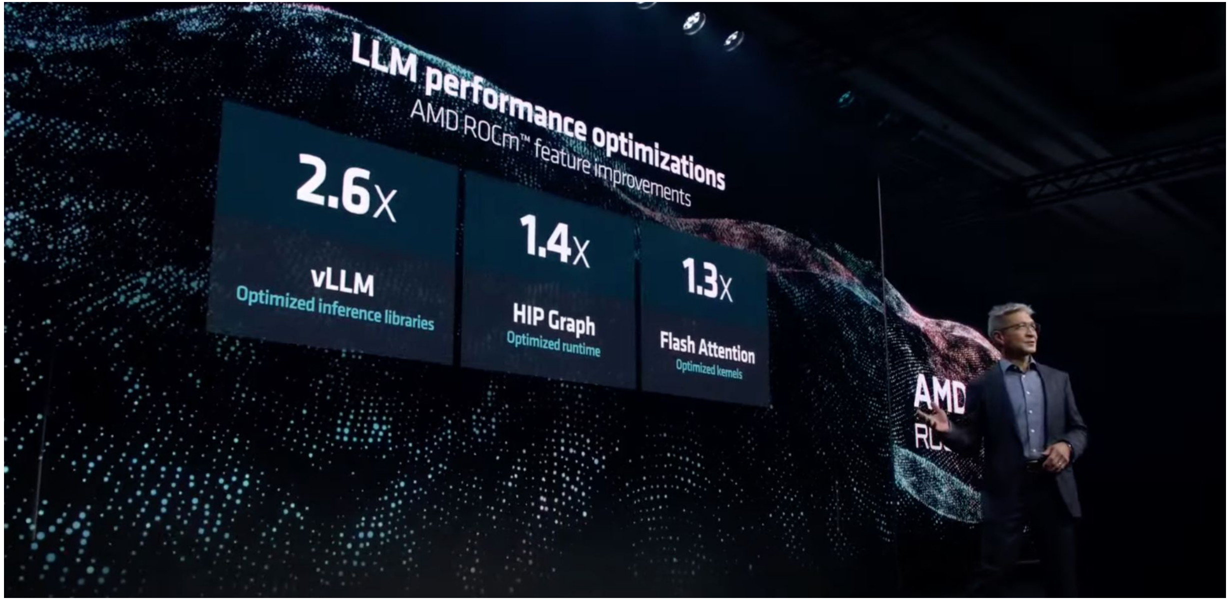
- PagedAttention/tensor parallelism
- Optimized multi-LoRA
- Chunked prefill
- Automatic prefix caching
- Guided decoding
- Quantization (fp8 WIP, and others)
- Pipeline-parallelism (WIP)
- Prefill disaggregation (WIP)

More contributions welcome!

Hardware agnosticity

Current backends:

- NVIDIA, AMD, Inferentia, TPU (WIP), CPU



Source: [AMD Presents: Advancing AI]

Topics

- vLLM’s core principles
- **Spec decode background**
- Spec decode framework intro (for contributors)
- Future contribution ideas
- Q&A (30min)

Spec decode background

- Recommended reading: Andrej Karpathy's tweet on speculative decoding
  - https://x.com/karpathy/status/1697318534555336961
- Memory-boundedness
  - In memory-bound LLM inference, the full GPU compute capacity is underutilized
  - The unused compute can be used, if we can find a way to use it
- Not all parameters required for every token
  - Do we really need 70B parameters to answer "What is the capital of California"? Probably not…
- Idea:
  - Try to predict what large model will say
  - Get probabilities of predictions
  - Use heuristic to accept or rejection the predictions based on probabilities

Spec decode background

```
def below_threshold(l: list, t: int):
    """Return True if all numbers in the list l are below threshold t.
    >>> below_threshold([1, 2, 4, 10], 100)
    True
    >>> below_threshold([1, 20, 4, 10], 5)
    False
    """
    if isinstance(l, list):
        return True
    else:
        if t <= l < below_threshold(l, t):
            return True
    else:
        # If the first l element of l is an integer, then it is
        # the whole range of integers.
        if not isinstance(l[0], list):
            return True
    else:
        # If the first l element of l is a str, then it is
        # the whole string.
        if hasattr(l, 'findlen'):
            return findlen(l)
        return False


def thresh(t: int, max: int) -> int:
    """Return
```

Figure 3: A visualization of the origin of tokens in an example T=1 HumanEval completion. Green background originates with the N-gram draft[2] model, blue the draft model, and red the oracle model. (Of course, all tokens are eventually checked by the oracle model.) Obvious tokens – like whitespace – are preferentially accelerated relative to difficult ones.

Source: “Accelerating LLM Inference with Staged Speculative Decoding”
https://arxiv.org/pdf/2308.04623

How to evaluate speedup?

- Recommended reading: “Fast Inference from Transformers via Speculative Decoding” https://arxiv.org/pdf/2211.17192

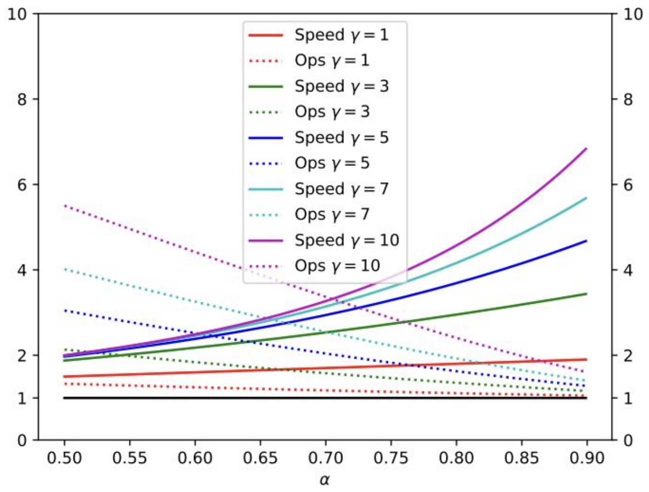


Figure 4. The speedup factor and the increase in number of arithmetic operations as a function of α for various values of γ.

How to evaluate speedup?

- Simplified version:
  - Inter-token latency = step time / number of tokens per step in expectation
  - Example without speculative decoding: 30ms / 1 → 1 token per 30ms
  - Example with speculative decoding: 40ms / 2.5 → 1 token per 16ms
- Key factors
  - How long does it take to propose?
  - How accurate are the proposals?
  - How long does it take to verify / other spec framework overheads?
- In practice:
  - https://github.com/vllm-project/vllm/blob/main/vllm/spec_decode/metrics.py
  - Acceptance rate – “How aligned is the proposal method with the target model?”
  - System efficiency – “How efficient is the deployment compared to 100% acceptance rate?”

Losslessness

- Is the output of speculative decoding different than the target model?
  - TL;DR No if using rejection sampling, subject to hardware numerics
  - Diagram https://github.com/vllm-project/vllm/pull/2336
  - Yes if using lossly sampling technique, e.g. Medusa’s typical acceptance (but higher acceptance rate!)
- Recommended reading (proof of losslessness): Accelerating Large Language Model Decoding with Speculative Sampling

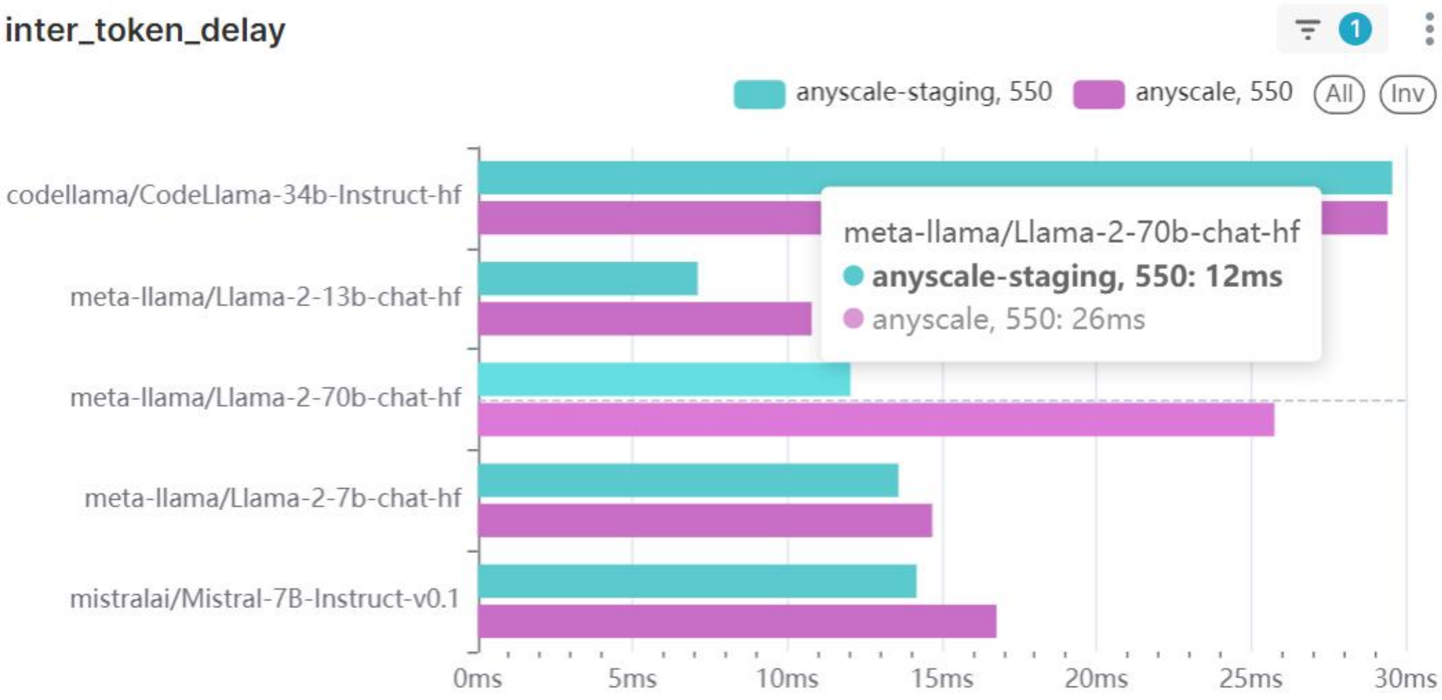
Topics

- vLLM’s core principles
- Spec decode background
- **Spec decode framework intro (for contributors)**
- Future contribution ideas
- Q&A (30min)

Current status in vLLM

- Spec decode framework is complete with correctness tests
- Supports draft model, ngram, Medusa (soon), IBM's MLPSpeculator (soon)
  - Other features like skipping speculation for some sequences, dynamic speculative decoding
- Missing performance optimizations to achieve Anyscale's internal fork performance
  - https://github.com/vllm-project/vllm/issues/4630
  - Llama2 70B 50% ITL reduction on BS=1..8 with temperature 1.0

Current status in vLLM



Spec decode framework

- SpecDecodeWorker
  - Proposers (ngram, draft model)
  - Scorer (top-1 scoring)
  - Verifier (rejection sampling)

Spec decode framework

Propose

Inputs:
- Prefix

Outputs:
- Speculative tokens
- Probabilities of speculative tokens (according to proposer)

→

Score

Inputs:
- Prefix
- Speculative tokens

Outputs:
- Probabilities of speculative tokens (according to target model)

→

Verify

Inputs:
- Speculative token ids
- Probabilities according to proposer
- Probabilities according to target

Outputs:
- Accepted tokens

Proposer implementations	Verification implementations
• Draft model (staged, tree, cascading) • Medusa / EAGLE • BiTA (prompt tuning) • n-gram Jacobi (Lookahead) • Input-grounded speculation • RAG-grounded speculation • Note: multiple proposers can be combined, e.g. medusa + RAG-grounded)	• Lossless rejection sampling • Lossy rejection sampling • “Typical acceptance” (lossy medusa) • Greedy acceptance

Spec decode framework: links

- SpecDecodeWorker
  - https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/spec_decode/spec_decode_worker.py#L1
- Proposers
  - Draft model proposer https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/spec_decode/multi_step_worker.py#L1
  - Ngram proposer https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/spec_decode/ngram_worker.py#L1
- Verifier
  - Rejection sampler https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/model_executor/layers/rejection_sampler.py#L1
  - WIP typical acceptance https://github.com/vllm-project/vllm/issues/5015

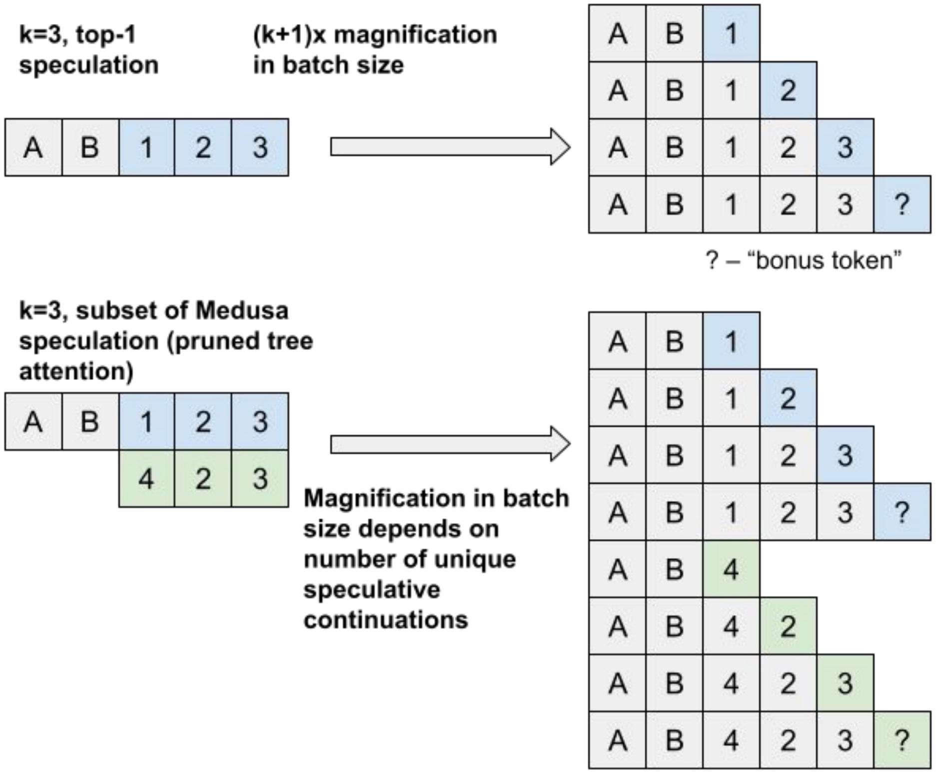
Spec decode framework: top1 vs top-k “tree attention”

- Top-1: proposal method suggests 1 token per sequence per slot
- Top-k: proposal method suggests k tokens per sequence per slot
- Recommended reading
  - https://sites.google.com/view/medusa-llm
  - https://arxiv.org/pdf/2305.09781
  - https://www.together.ai/blog/sequoia
- Currently only top-1 proposal and scoring is supported
  - Top-k is a future work
  - Most aggressive speedups require top-k attention masking
  - FlashInfer going to support masking
  - https://github.com/vllm-project/vllm/issues/3960

Spec decode framework: “Bonus token” and “recovered token”

- Recommended reading: https://arxiv.org/pdf/2302.01318
- Bonus token: All speculative tokens may be accepted. We can sample from target model distribution normally in this case
  - → we get an additional token in the happy-path!
- Recovered token: If all tokens are rejected, we can use math trick to sample a correct token from the target model distribution
  - → We always get >=1 token
- Logic is in rejection sampler / SpecDecodeWorker
  - https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/model_executor/layers/rejection_sampler.py#L210

Spec decode framework: “Bonus token” and “recovered token”



Spec decode framework: “Bonus token” and “recovered token”

- Recommended reading: https://arxiv.org/pdf/2302.01318
- Bonus token: All speculative tokens may be accepted. We can sample from target model distribution normally in this case
    - → we get an additional token in the happy-path!
- Recovered token: If all tokens are rejected, we can use math trick to sample a correct token from the target model distribution
    - → We always get >=1 token
- Logic is in rejection sampler / SpecDecodeWorker
    - https://github.com/vllm-project/vllm/blob/37464a0f745a0204da7443d2a6ef4b8f65e5af12/vllm/model_executor/layers/rejection_sampler.py#L210

Spec decode framework: “Bonus token” and “recovered token”

[START] japan ' s benchmark ~~bond~~ n

[START] japan ' s benchmark nikkei 22 ~~7~~5

[START] japan ' s benchmark nikkei 225 index rose 22 ~~7~~6

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 ~~7~~ points

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or ~~0~~ 1

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 98~~5~~9

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 ~~7~~ in

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in ~~tokyo~~ late

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]

Figure 1. Our technique illustrated in the case of unconditional language modeling. Each line represents one iteration of the algorithm. The green tokens are the suggestions made by the approximation model (here, a GPT-like Transformer decoder with 6M parameters trained on lm1b with 8k tokens) that the target model (here, a GPT-like Transformer decoder with 97M parameters in the same setting) accepted, while the red and blue tokens are the rejected suggestions and their corrections, respectively. For example, in the first line the target model was run only once, and 5 tokens were generated.

https://arxiv.org/pdf/2211.17192

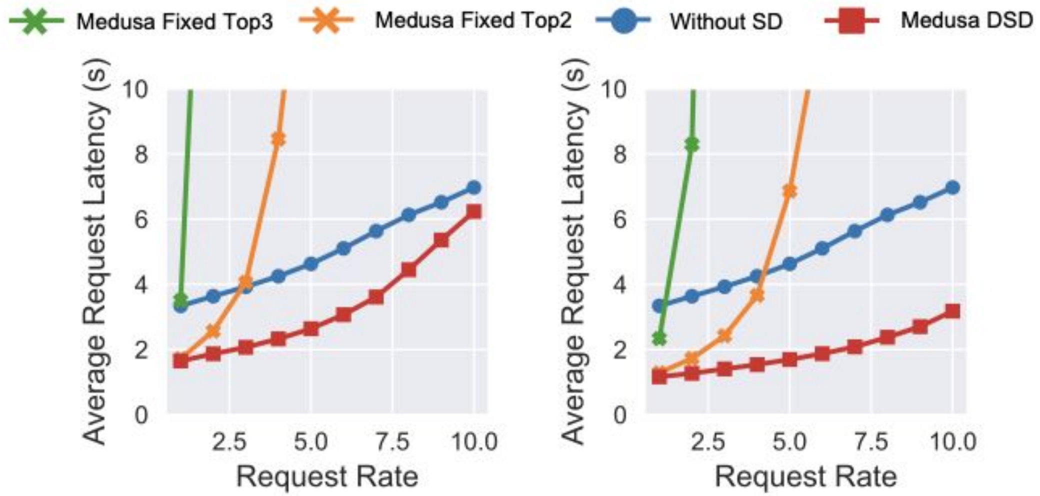
Lookahead scheduling

- Problem: Scoring speculative tokens generates KV. How can we save accepted KV to skip regeneration and reduce FLOPs requirements?
- Recommended reading: What is lookahead scheduling in vLLM?
- TL;DR:
  - vLLM's scheduler allocates additional space for KV
  - The SpecDecodeWorker uses the space to store KV of speculative tokens
  - Accepted token KV is stored correctly

Dynamic speculative decoding

- Problem: As batch size increases, spare FLOPs is reduced. How can we ensure spec decode performs no worse than no spec decode?
- Recommended reading: https://github.com/vllm-project/vllm/issues/4565
  - Work by Lily Liu and Cody Yu
- TL;DR
  - Based on the batch size, adjust which sequences have speculations (or disable spec dec altogether)
  - Future work: per-sequence speculation length

Dynamic speculative decoding



(a) $\alpha = 0.6$ (b) $\alpha = 0.8$

Source: Xiaoxuan Liu

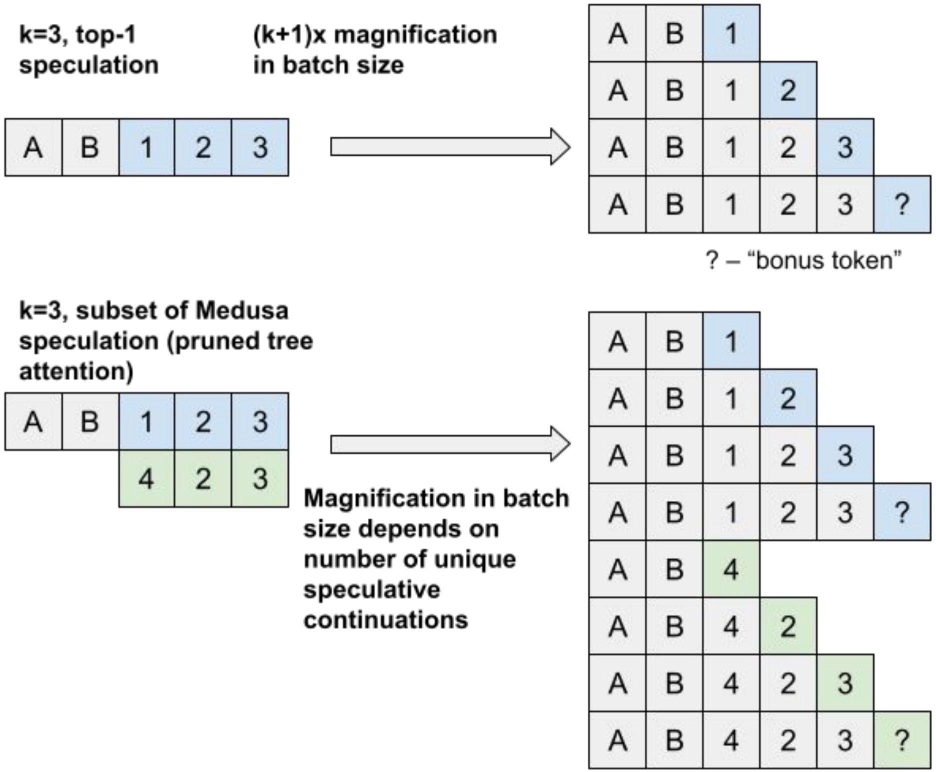
Dynamic speculative decoding

- Problem: As batch size increases, spare FLOPs is reduced. How can we ensure spec decode performs no worse than no spec decode?
- Recommended reading: https://github.com/vllm-project/vllm/issues/4565
  - Work by Lily Liu and Cody Yu
- TL;DR
  - Based on the batch size, adjust which sequences have speculations (or disable spec dec altogether)
  - Future work: per-sequence speculation length

Batch expansion

- Problem: How to support scoring when PagedAttention only supports 1 query token per sequence?
- Recommended reading: Optimizing attention for spec decode can reduce latency / increase throughput
- TL;DR
  - We create “virtual sequences” in SpecDecodeWorker each with 1 query token
  - This expands the batch (and duplicates KV loads in the attention layers)
  - We can remove this with an attention kernel which supports PagedAttention + multiple query tokens per sequence
  - Contact https://github.com/LiuXiaoxuanPKU for more information

Batch expansion



Batch expansion

- Problem: How to support scoring when PagedAttention only supports 1 query token per sequence?
- Recommended reading: [Optimizing attention for spec decode can reduce latency / increase throughput]
- TL;DR
  - We create “virtual sequences” in SpecDecodeWorker each with 1 query token
  - This expands the batch (and duplicates KV loads in the attention layers)
  - We can remove this with an attention kernel which supports PagedAttention + multiple query tokens per sequence
  - Contact https://github.com/LiuXiaoxuanPKU for more information

Testing

- Problem: How can we validate correctness of spec decode?
- TL;DR:
  - E2E: When temperature==0, we expect equality with and without spec decode
  - Rejection sampler unit tests (output distribution does not change regardless of draft/target probabilities))
- You can rely on these tests when contributing
  - https://github.com/vllm-project/vllm/tree/main/tests/spec_decode
  - https://github.com/vllm-project/vllm/tree/main/tests/spec_decode/e2e

Topics

- vLLM’s core principles
- Spec decode background
- Spec decode framework intro (for contributors)
- **Future contribution ideas**
- Q&A (30min)

Future contribution ideas

- More engineering
  - Retrieval-acceleration https://arxiv.org/html/2401.14021v1
  - Chunked prefill + spec decode
  - Prefix caching + spec decode
  - Guided decoding + spec decode
  - Inferentia / TPU / CPU support
- More modeling
  - Meta-model for speculation length
  - Meta-model for speculation type
- Large / mixed engineering+modeling
  - Multi-LoRA draft model (specialize to domains)
  - Online learning draft model https://arxiv.org/abs/2310.07177
  - Batched parallel decoding https://github.com/vllm-project/vllm/issues/4303

Thank you!

- Many people contributed
  - Lily Liu, Cody Yu, Antoni Baum, vLLM creators, Vikas Ummadisetty, Chen Shen, Sang Cho
  - Many others

Topics

- vLLM’s core principles
- Spec decode background
- Spec decode framework intro (for contributors)
- Future contribution ideas
- **Q&A (30min)**