SpecAdapt: Adaptive Semantic Prompt Routing for Speculative Decoding

1st Aakash Aanegola
Columbia University
aa5506@columbia.edu

2nd Nicholas Bykhovsky
Columbia University
nb3227@columbia.edu

3rd Aryamaan Saha
Columbia University
as7482@columbia.edu

***Abstract*—We propose a modular framework to optimize language model inference through adaptive speculative decoding. The first stage involves an extensive empirical evaluation of five decoding strategies - Medusa, EAGLE, fuzzy speculative decoding, standard draft-and-verify, and a non-speculative baseline - measured across multiple dimensions including throughput, token-level latency, and consistency with non-speculative outputs. Based on these benchmarks, the second stage introduces a prompt-aware routing mechanism that embeds incoming queries via a sentence embedding model and maps them to clusters derived from prompt semantics. Each cluster is assigned the speculative decoding method that yielded the best performance for that region in the prompt representation space. This approach enables dynamic selection of decoding strategies conditioned on prompt semantics, yielding substantial runtime improvements while preserving output quality (1.63x runtime improvement, over the average 1.51-1.53 observed empirically).**

***Index Terms*—Large Language Models, Speculative Decoding, Prompt-aware Optimization, Inference Acceleration**

I. Introduction

Recent advances in speculative decoding have enabled significant acceleration of language model inference by allowing parallel or approximate token generation. Methods like Medusa [1], EAGLE [2], fuzzy speculative decoding [3], and draft-and-verify [4] introduce various mechanisms to reduce latency while maintaining generation fidelity. These techniques are gaining traction in production settings where inference costs and throughput are critical, particularly for applications built on large language models (LLMs) deployed in large-scale, multi-tenant environments.

Benchmarks like SpecBench [5] have systematically evaluated speculative decoding strategies across a broad spectrum of tasks—ranging from arithmetic reasoning to code generation—highlighting the trade-offs between speed and accuracy. However, these benchmarks primarily focus on task type or dataset as the unit of analysis, without considering the semantic properties of individual prompts. This overlooks an important observation: the effectiveness of a speculative decoding strategy often depends not just on the task, but on the semantic structure and complexity of the prompt itself.

In high-throughput inference settings, where data centers serve a heterogeneous stream of prompts across domains and user intents, using a single decoding strategy may leave performance gains on the table. Instead, we hypothesize that prompt semantics—as captured by sentence embeddings—offer a lightweight and effective signal for online routing to the most suitable speculative decoding method. Unlike task classification, this embedding-based approach is model-agnostic and deployable with minimal overhead.

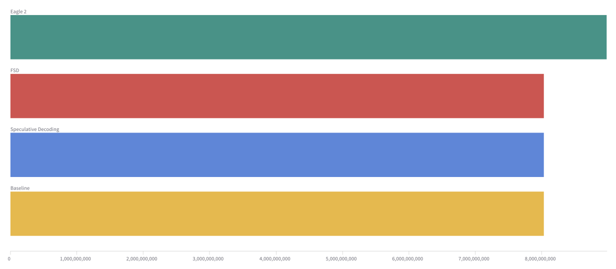
We present a two-stage framework to validate this idea. First, we extend prior work by benchmarking several speculative decoding strategies (Medusa [1], EAGLE [2], fuzzy decoding [3], draft-and-verify [4], and a baseline) across diverse prompt types, quantifying their speed and accuracy trade-offs. Then, we develop a routing mechanism that embeds incoming prompts, clusters them using K-Means, and assigns each cluster the decoding method that performed best during benchmarking. Our experiments show that this approach leads to consistent runtime improvements compared to using any single strategy in isolation. To the best of our knowledge, we are the first system that incorporates prompt semantics to guide speculative-decoding method selection.

II. Related Work

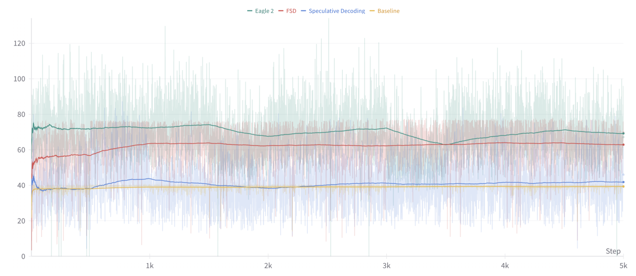
Speculative decoding aims to accelerate language model inference by generating multiple candidate tokens using a fast draft model and then verifying them using the full target model. The standard draft-and-verify method introduced by Leviathan et al. [4] executes this process in a greedy, token-by-token manner, with a fixed-length draft followed by strict verification.

EAGLE (Efficiently Accelerated Generation via Learning to Estimate) [2] builds on this idea by introducing a confidence-aware verification policy. Instead of performing full verification for every token, EAGLE learns to predict the likelihood that a draft token will be accepted, using a lightweight estimator network trained alongside the main model. This allows EAGLE to selectively skip verification steps for high-confidence tokens, significantly improving throughput while maintaining generation fidelity. EAGLE also supports speculative decoding with dynamic step sizes, making it adaptable to prompt complexity.

In contrast, Fuzzy Speculative Decoding [3] adopts a probabilistic relaxation of verification. Rather than requiring exact match between the draft and target model token predictions, fuzzy decoding introduces a fuzzy acceptance criterion based on top-k similarity or KL divergence thresholds. This approach increases the acceptance rate of speculative tokens, which reduces fallback costs and improves performance in smoother



(a) Parameter count across methods.



(b) Average tokens per second. Note that all techniques have high variance, which helped us hypothesize that there may be technique-specific high-performance subspaces.

Fig. 1: Comparison of decoding methods across model size and generation throughput for the ultrachat dataset with 5k input prompts.

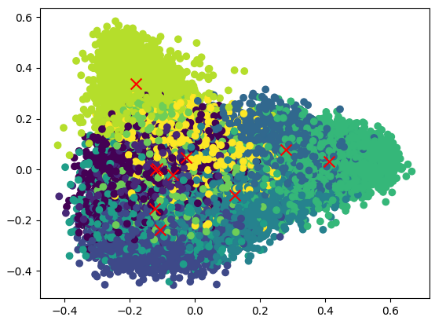


Fig. 2: Clusters and centroids for 5,000 prompt embeddings, projected to two dimensions using PCA. The separation between clusters suggests meaningful semantic structure in the input space, supporting the use of clustering as a basis for decoding strategy assignment.

token distributions. However, fuzzy acceptance can also propagate low-confidence tokens in high-entropy regions, leading to generation quality degradation in some cases.

Another complementary method medusa [1] proposes training heads that predict the i^{th} token from the current position and come up with multiple proposals in parallel that can be verified in parallel - the probability of identifying a continuation this way is relatively high but it comes with the additional overhead of training the new heads from the last hidden layer.

All three methods represent significant steps beyond standard draft-and-verify: EAGLE optimizes the verification process, while Fuzzy Speculative Decoding (FSD) modifies the acceptance condition itself and Medusa modifies the drafting phase. Despite these advances, they are still typically deployed statically, without adapting to the semantic nature of the input. Our work addresses this by dynamically routing prompts to the most appropriate decoding strategy based on semantic embeddings—yielding consistent speedups while respecting the strengths and weaknesses of each method.

Benchmarking frameworks such as SpecBench [5] provide systematic comparisons of speculative decoding techniques across a range of tasks (e.g., reasoning, summarization, dialogue). However, these evaluations typically aggregate performance over datasets or tasks, without examining how different decoding methods behave under varied prompt semantics. Our results show that speculative methods exhibit high performance variance across prompt types, and that routing prompts based on sentence embeddings enables consistent speedups beyond what any single method achieves alone. For example, we find that EAGLE and FSD struggle on prompts that encourage creativity while standard draft-and-verify performs well - this isn't task specific but rather semantic specific.

Recent work in input-aware inference explores ways to adapt model behavior based on input characteristics—examples include early exiting, Mixture-of-Experts routing [6], and conditional decoding. Systems like DeepSpeed-Inference [7] implement dynamic layer skipping and runtime pruning to reduce latency. However, these approaches often require architectural modifications or supervised training objectives. In contrast, our method uses unsupervised clustering of lightweight sentence embeddings, enabling prompt-aware routing without fine-tuning or changes to the base model architecture.

III. Methods

A. Experimental Setup

All experiments were conducted on NVIDIA L40 GPUs with 40 GB of VRAM. Each speculative decoding method was assigned its own dedicated GPU to simulate a multi-strategy deployment environment and eliminate cross-method interference. Due to memory constraints and the need to fit models fully on-chip, we use the LLaMA 3.1B and 8B variants as our base models. All models were hosted using PyTorch eager mode unless otherwise noted.

We evaluated decoding strategies on a corpus of 30,000 prompts from the UltraChat dataset. Of these, 5,000 prompts were used for clustering and speedup profiling. Prompt embeddings were generated using the all-MiniLM-L6-v2 sentence encoder, yielding 384-dimensional vectors. K-means clustering (with $k = 10$) took approximately 1 minute to complete on this dataset.

At evaluation time, prompts were routed to decoding strategies based on their nearest cluster centroid. Each cluster contained approximately 3,000 prompts on average (standard deviation: 682). Runtime metrics, including total generation time and tokens per second, were logged to Weights and Biases for analysis.

Speedup figures are reported relative to naive decoding using standard PyTorch, and we visualize variance across decoding strategies in Figure 1b. Notably, all methods exhibit high token-level variance, motivating the use of routing to mitigate performance instability across heterogeneous prompts.

B. Benchmarking

We conducted extensive profiling and benchmarking of several speculative decoding methods across different inference engines to characterize the conditions under which each technique delivers optimal performance. Our initial focus was on low-level execution efficiency: we used the PyTorch profiler to analyze GPU-level behavior, including kernel launches, memory operations, and operator fusion.

One of our early objectives was to investigate whether writing custom CUDA kernels could yield meaningful speedups in the speculative decoding pipeline. However, our profiling revealed that existing libraries—particularly Hugging Face Transformers—already employ a high degree of kernel- and graph-level optimization. As a result, there is limited headroom for additional gains through custom kernel development that would generalize across decoding strategies.

In the sections that follow, we detail our profiling methodology and findings, comparing PyTorch's eager mode with optimized inference frameworks such as vLLM. As shown in Fig. 2, running inference using vLLM yields noticeably higher and more stable GPU utilization compared to standard PyTorch. These insights inform the architectural decisions behind our dynamic routing framework, motivating a shift in focus from kernel-level optimization to high-level scheduling and prompt-aware inference selection.

Metric	**Value**
Silhouette Score (sampled)	0.1463
Davies-Bouldin Index	4.6091
Calinski-Harabasz Index	124.7877
Inertia	2763.05
Average Cluster Size	500.0 ± 107.09

TABLE I: Clustering metrics for 5,000 prompt embeddings using K-Means (k = 10).

1) Torch Profiler: We used the PyTorch profiler to capture detailed GPU-level statistics, including kernel launches, memory operations, and execution timelines. These traces were manually analyzed to identify potential optimization opportunities. Unsurprisingly, however, our findings confirmed that Hugging Face Transformers already apply aggressive low-level optimizations, leaving limited room for further kernel-level improvements.

2) vLLM: We profiled all the decoding methods on the vLLM inference engine [8], a high-throughput system designed for efficient KV cache memory management. vLLM introduces PagedAttention, which decouples logical and physical KV cache allocations using a virtual memory-style paging system. This architecture eliminates internal and external memory fragmentation, enabling significantly higher batch sizes during inference, especially for long sequences. Profiling was conducted using a custom script that iterates over prompt clusters and logs runtime statistics, such as total time and tokens-per-second, to WandB.

Method	**Harmonic Mean Speedup**
Spec-Dec (vLLM)	**1.87**
Eagle1 (vLLM)	1.81
Spec-Dec (Std)	1.52
FSD (Std)	1.34
Eagle2 (Std)	1.33
Naive (vLLM)	1.13

TABLE II: Harmonic mean speedups across all clusters for each decoding method, relative to naive decoding on standard PyTorch.

C. Clustering

To enable prompt-specific decoding strategies without incurring inference-time overhead, we adopt a lightweight offline clustering approach. Prompts are embedded using the all-MiniLM-L6-v2 sentence encoder, and the resulting 384-dimensional vectors are clustered using K-Means. This unsupervised procedure groups semantically similar prompts and forms the basis for assigning an optimal speculative decoding strategy to each cluster.

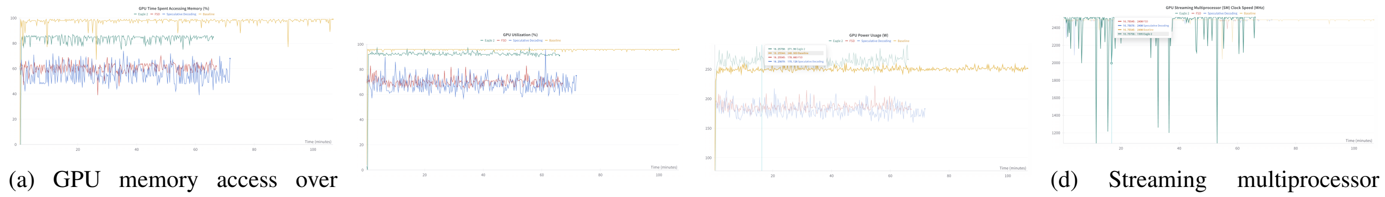
Decoding strategies are selected per cluster using speedup data collected during benchmarking. For each prompt, we record the speedup achieved by different decoding techniques and compute the harmonic mean within each cluster to reflect overall runtime efficiency while reducing sensitivity to outliers.

The clustering model is trained on a 5,000-sample subset of a 30,000-prompt corpus (UltraChat) and evaluated on the full dataset. As shown in Fig.2, the 2D PCA projection reveals visible semantic separation qualitatively. TableI reports quantitative metrics, showing reasonably compact and distinct clusters. While not optimal, the clustering is sufficient to guide decoding decisions, and more sophisticated methods could improve separation at the cost of additional overhead which may hurt the overall speedup.

Although the current setup is static, the approach is extensible to an online setting, where cluster centers can be updated or refined post-hoc based on observed speedups. This provides a scalable foundation for deploying prompt-aware speculative decoding in systems with evolving prompt distributions.

D. Semantic Routing

At inference time, each incoming prompt is passed through the all-MiniLM-L6-v2 sentence encoder to generate a 384-



(a) GPU memory access over time. (b) GPU utilization (%). (c) Power usage (Watts). (d) Streaming multiprocessor (SM) utilization.

Fig. 3: GPU-level performance metrics collected during speculative decoding execution. These include memory bandwidth, utilization, power draw, and SM activity, each offering insight into the system bottlenecks under different decoding strategies.

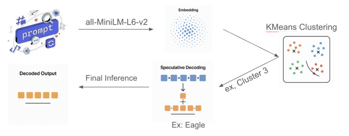


Fig. 4: A graphical representation of the routing algorithm. Every component is easily replaced, and can be performed in an online fashion.

dimensional embedding. This embedding is then assigned to the nearest cluster centroid using a pre-trained K-Means model. The process introduces negligible latency—embedding a single prompt takes approximately 11 ms on average, and label prediction is near-instantaneous. For high-throughput scenarios, both steps can be trivially parallelized across batches.

Each cluster is associated with a single decoding strategy, pre-selected based on offline benchmarking of speedup and accuracy trade-offs. The selected speculative decoding method is then invoked to generate the output. As illustrated in Fig. 4, this process enables dynamic strategy selection based on semantic characteristics of the prompt.

To support multi-strategy inference, we implement routing as a separate inference gateway that runs independently of the model servers. In our prototype, each decoding method is loaded onto a dedicated GPU to simulate a multi-tenant environment typical of large-scale deployments. Decoding methods are accessed via API calls, but our architecture is extensible to a unified custom scheduling layer capable of handling load balancing and warm-started model switching.

We also build a chatbot interface using the same routing system to demonstrate its responsiveness in a user-facing application. Since MiniLM can embed over 14,000 prompts per minute, the added latency is effectively negligible even in interactive settings.

While our current system uses static cluster-to-method mappings, it can be extended to a dynamic variant: for example, by updating strategy assignments based on real-time throughput metrics or adapting to drift in the prompt distribution. This opens the door to an online learning framework where the system continuously optimizes decoding policy without retraining the underlying models.

IV. Results

A. *Profile Results*

From Fig.3a to Fig.3d, several key observations emerge about the underlying hardware behavior of the decoding strategies we benchmarked. The SM clock speed plots reveal that speculative decoding methods experience periodic dips not seen in the baseline, which maintains a more stable execution profile. These dips likely correspond to the intermittent gap between draft generation and target model verification, where the GPU may be partially idle waiting for speculative rollouts to complete.

The power utilization metrics (Fig. 3c) further highlight an efficiency tradeoff. While Eagle 2 delivers the highest throughput overall, it does so at the cost of increased power consumption. In contrast, fuzzy speculative decoding (FSD) and naive speculative decoding achieve comparable or superior runtime performance with significantly lower power usage, making them more favorable for power-constrained deployments.

Memory behavior offers additional insights. As shown in Fig. 3a, Eagle 2 exhibits the highest memory bandwidth usage, whereas the baseline requires the least. This is expected, as speculative methods must load and maintain both draft and target models simultaneously, increasing memory pressure. Interestingly, the baseline method also demonstrates the poorest memory access performance—indicating a memory-bound bottleneck that speculative decoding partially mitigates by increasing arithmetic intensity. Eagle 2, in particular, stands to benefit from further reductions in memory access latency,

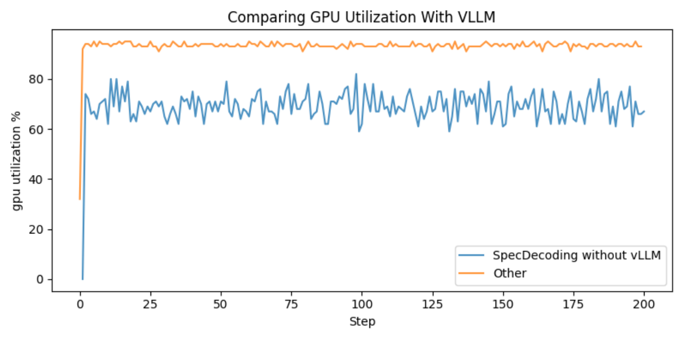


Fig. 5: Comparing GPU Utilization With vLLM

Cluster	FSD (Std)	Spec-Dec (Std)	Eagle2 (Std)	Naive (vLLM)	Spec-Dec (vLLM)	Eagle1 (vLLM)
0	1.51	1.53	**1.84**	1.13	1.73	1.74
1	1.60	1.83	1.58	1.13	2.13	**2.20**
2	1.66	1.75	1.71	1.14	**1.86**	1.77
3	1.60	1.37	1.36	1.13	1.74	**1.76**
4	0.52	1.24	0.52	1.13	**1.83**	1.80
5	1.73	1.52	1.78	1.13	**1.91**	1.83
6	1.58	1.51	1.27	1.13	**1.88**	1.74
7	1.80	1.51	1.73	1.13	**1.93**	1.86
8	1.59	1.58	1.71	1.13	**1.92**	1.78
9	1.63	1.54	1.66	1.13	**1.82**	1.75

TABLE III: Cluster-wise speedup comparison across decoding strategies and inference engines. All speedups are measured relative to a baseline of naive decoding on standard PyTorch inference. Best values per cluster are shown in bold.

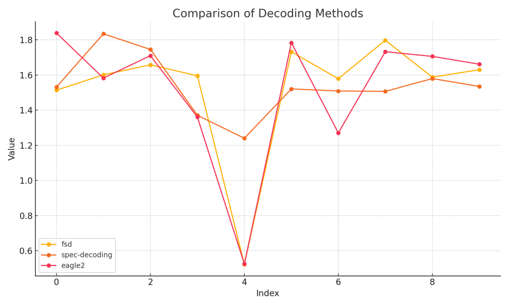


Fig. 6: Comparing performance of different speculative decoding techniques across clusters. As we can see fuzzy speculative decoding (fsd) and eagle 2 (eagle2) generally outperform the naive speculative decoding technique but there's some clusters on which it beats them (especially cluster 4).

which could close the gap between its raw speed and power efficiency.

GPU memory allocation patterns were consistent across all methods: a sharp ramp-up during initialization followed by a steady state proportional to model size, as expected given the static nature of LLM parameter footprints.

Perhaps most surprising are the findings from Fig. 3d, which show that speculative decoding techniques exhibit higher idle time than the baseline. This counterintuitive result may be due to LLaMA's highly optimized single-model inference path. In speculative setups, overhead from context switching or poor pipeline parallelism could cause stalls that underutilize the GPU. These inefficiencies point to an opportunity for architectural enhancements: a multi-GPU or heterogeneous deployment where the draft and target models are executed on separate accelerators could improve SM utilization and overall throughput—a direction we identify as promising future work.

1) Token-level statistics: Fig. 1b presents a comparative analysis of speculative decoding methods across model size, tokens per second, and total generation time. Although all methods share the same base model, Eagle 2 has a slightly higher parameter count, likely due to auxiliary components introduced for drafting and verification. The tokens per second plot shows Eagle 2 achieving the highest average throughput, followed by FSD, with naive speculative decoding trailing behind—frequently falling below the baseline in performance. These trends are echoed in the generation total time plot, where both Eagle 2 and FSD consistently outperform the baseline and speculative decoding, albeit with considerable variance.

Importantly, all methods display high step-to-step variance, suggesting that performance fluctuates significantly depending on the nature of the prompt. This observation, coupled with the fact that the mean accepted tokens (MAT) metric is largely compensated for during verification, reinforces the idea that no single decoding method is universally optimal. These insights directly motivate our routing strategy, which leverages prompt embeddings to dynamically assign the most suitable decoding method at inference time—achieving more stable performance across heterogeneous input distributions.

2) vLLM: We found that naive (non-speculative decoding) on vLLM provides a modest speedup of 1.13x compared to naive decoding on a standard PyTorch engine, due to backend-level optimizations discussed earlier.

When applying speculative decoding, we observe even greater gains. Draft-and-verify speculative decoding on vLLM achieves the highest average performance across all methods, with a harmonic mean speedup of 1.87× over the baseline. This outperforms the same speculative decoding strategy on a standard engine (harmonic mean: 1.52×). Table II summarizes the average performance of each method, while Table III provides a per-cluster breakdown.

EAGLE1 on vLLM is a close competitor, achieving a harmonic mean speedup of 1.81×, but underperforms slightly compared to draft-and-verify. Upon investigation, we attribute this to implementation discrepancies between the vLLM adaptation and the original EAGLE design. Notably, the vLLM implementation introduces additional input and output layer normalization steps and lacks the top-k tree-based verification kernel described in the original paper. These differences result in lower draft token acceptance rates and introduce additional runtime overhead, which ultimately limits performance. [9]

Despite these issues, EAGLE1's strong performance suggests that a faithful reproduction of the original implementation within the vLLM architecture could yield even higher gains. These findings reinforce the idea that inference acceleration can be significantly amplified when model-level strategies (such as speculative decoding) are paired with optimized

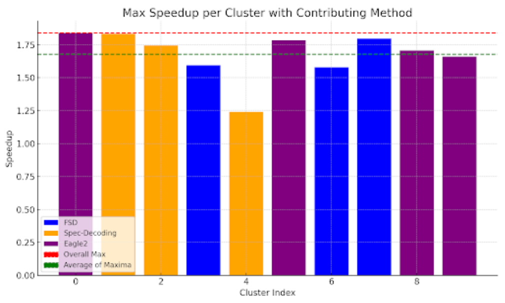


Fig. 7: Indicating the technique that was selected for each cluster along with the corresponding speedup. We also show the theoretical bound and the average of the maxima.

inference engines, rather than applied in isolation.

B. Clustering and Routing

We re-evaluate speculative decoding strategies based on their cluster-specific performance, rather than relying solely on their overall average speedup. This allows us to retain methods like standard draft-and-verify, which performs well on specific clusters (notably cluster 4) where other techniques underperform—sometimes achieving only 0.5× the baseline speed. Using this routing strategy, we achieve a 1.63× overall speedup, outperforming any individual decoding method, the best of which achieves 1.52×. Notably, we exclude Medusa from this comparison due to the absence of publicly available weights for LLaMA 3.1 8B, which we selected as our baseline model after evaluating Gemma, LLaMA, and Mistral.

As shown in Fig. 6, speculative decoding techniques exhibit high performance variance across prompt types. Cluster 4, in particular, contains creative prompts such as "Write a dialogue between two high school students preparing for prom" and "Create a mood board for a travel campaign." Here, aggressive methods like EAGLE and fuzzy speculative decoding underperform, achieving less than 0.5× baseline speed due to frequent token rejections in high-entropy generations. In contrast, the standard draft-and-verify strategy excels, benefiting from its conservative verification mechanism that better preserves coherence in open-ended outputs.

Fig. 7 highlights that each decoding method achieves peak performance in different clusters. This cluster-level specialization supports our core hypothesis: routing prompts to decoding methods based on their semantic properties yields consistent runtime improvements at negligible cost, thanks to the lightweight sentence embedding step and precomputation of cluster centers.

C. Choosing the number of clusters

We also demonstrate performance across different k (number of clusters) and empirically find k=10 to provide the best speedup as seen in Table IV. We also include the cluster-specific speedups for k=10 in Table III.

TABLE IV: Comparison of harmonic mean speedup across different cluster sizes.

Clusters (k)	**Harmonic Mean Speedup**
5	1.6009
10	1.6576
20	1.6271

V. Conclusion and Future Work

We introduced a prompt-aware speculative decoding framework that dynamically selects the most suitable decoding strategy based on the content of incoming prompts. By embedding prompts and clustering them in a latent space, we enable a decoding strategy assignment that empirically yields the best runtime performance within each cluster. Our evaluation demonstrates that this approach achieves a throughput improvement of 1.63× over baseline, compared to 1.52× when using the static best-performing decoding method in our compute bound setup. The overhead introduced by embedding computation is nominal, and becomes increasingly negligible at inference scale and can be further improved by using lightweight encoder models.

These results suggest that prompt semantics can serve as an effective signal for optimizing inference. Rather than relying on a single decoding strategy across all inputs, our framework enables adaptive routing with minimal integration cost, offering a new degree of flexibility in system design. As speculative decoding techniques continue to evolve, we see this modular routing architecture as a foundation for future systems that incorporate adaptive, prompt-aware inference paths.

VI. Author Contributions

The contributions of the authors to this work are as follows:

Aakash Aanegola: Contributed to the initial research and basic benchmarking of speculative decoding methods, focusing on standard speculative decoding, fuzzy speculative decoding, and EAGLE2. He also led the prompt clustering and embedding efforts and was responsible for the implementation of the benchmarking and routing system.

Nico Bykhovsky: Contributed to the initial speculative decoding method research and basic benchmarking, focusing on Medusa and EAGLE2. He led the design of the routing system architecture.

Aryamaan Saha: Conducted initial method research and basic benchmarking related to the vLLM inference engine and implemented the EAGLE integration within the benchmarking framework. He was responsible for the implementation of the routing system, focusing on state management and user interface.

References

[1] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[2] Y. Li, F. Wei, C. Zhang, and H. Zhang, "EAGLE-2: Faster inference of language models with dynamic draft trees," in *Empirical Methods in Natural Language Processing*, 2024.

[3] M. Holsman, Y. Huang, and B. Dhingra, “Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff,” 2025. [Online]. Available: https://arxiv.org/abs/2502.20704

[4] Y. Leviathan, M. Kalman, and Y. Matias, “Fast inference from transformers via speculative decoding,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 274–19 286.

[5] H. Xia, Z. Yang, Q. Dong, P. Wang, Y. Li, T. Ge, T. Liu, W. Li, and Z. Sui, “Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding,” in *Findings of the Association for Computational Linguistics ACL 2024*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand and virtual meeting: Association for Computational Linguistics, Aug. 2024, pp. 7655–7671. [Online]. Available: https://aclanthology.org/2024.findings-acl.456

[6] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” 2022.

[7] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, “Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’22. IEEE Press, 2022.

[8] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, “Efficient memory management for large language model serving with pagedattention,” 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[9] X. Liu *et al.*, “[Performance]: vLLM Eagle performance is worse than expected,” https://github.com/vllm-project/vllm/issues/9565, 2024, gitHub Issue #9565, accessed May 10, 2025.