DynaDFlash: Dynamic Feature Projection for Block Diffusion Speculators

Aadi Kulshrestha
a7kulshrestha@uwaterloo.ca

Reese Chong
r25chong@uwaterloo.ca

November 12, 2026

Abstract

DFlash trains a parallel draft model on hidden features from its target model. Inspired by the previous work of DFlare that introduced per layer combinations of target hidden states, we test whether context dependent weighting of these states helps improve acceptance length of block diffusion speculators. We trained both a DFlash baseline for the Qwen 3 of block size 16, and a dynamic model that computes gates for each layer to affect the magnitude of hidden features before projection. While mean acceptance length between the dynamic model and the DFlash baseline didn't significantly differ, we notice that the dynamic model excels for drafting in the context of math. In an evaluation of 700 prompts from the MetaMathQA and Orca-Math datasets, the mean accepted length improves by 5%. We also found this lead to a 4.1% higher throughput for a subset of the same dataset in the context of speculative decoding.

1 Introduction

Speculative decoding uses a smaller draft model to predict tokens that are then verified by a larger target model [1, 2]. DFlash does this by training a block diffusion model to draft multiple tokens in parallel, using hidden states from the target model as additional context that is injected at each layer [5].

DFlare builds on this idea by learning different combinations of target hidden states for different draft layers [6]. However, these combinations remain fixed for each layer after training. We wanted to test whether allowing the weights to change depending on the input context could further improve drafting.

We modify DFlash by adding a small gate that computes weights for the target hidden states before they are projected. We then train both the original DFlash architecture and our modified version under the same conditions and compare their acceptance lengths and decoding throughput.

2 Background

Autoregressive LLMs generate tokens one at a time. Each token requires a new forward pass through the target model, which slows down generation. Speculative decoding allows us to produce more tokens for each forward pass of the target model. A smaller model first drafts several tokens, and the target model checks them together in a single forward pass [1, 2].

Under greedy decoding, a draft token is accepted if it matches the highest likelihood token the target would have produced. Tokens are accepted until the first mismatch, at which point the target produces the next token instead. The final output is therefore the same as ordinary greedy decoding. For sampling, speculative decoding uses a different acceptance formula to preserve the target model's output distribution [1, 2].

The speedup comes from verifying several tokens in parallel rather than running the target separately for every token. A useful approximation for average time per generated token is [5]

$$L \approx \frac{T_{\text{draft}} + T_{\text{verify}}}{\tau}, \tag{1}$$

where T_{draft} is the time it takes for the drafter to propose a block, T_{verify} is the time to check it via the target model, and τ is the average number of tokens produced per verification cycle (the accepted tokens + 1). Examining this approximation it becomes clear that to improve throughput we must balance the latency of the drafter with its acceptance length.

DFlash reduces drafting time by predicting the next 15 tokens in parallel from a block of masked positions, rather than generating each draft token one at a time [5]. The target then verifies the block. This makes both drafting time and acceptance length important when comparing draft models.

3 Models

DFlash takes hidden states from layers 2, 10, 18, 26, and 34 of the target model, concatenates them, and projects them into the draft model.

Let $\mathbf{h}_t^{(l)}$ be the hidden state from target layer l at token position t. DFlash combines these states using a learned projection W_c:

$$\begin{aligned}\mathbf{z}_t &= [\mathbf{h}_t^{(2)}; \mathbf{h}_t^{(10)}; \mathbf{h}_t^{(18)}; \\ &\quad \mathbf{h}_t^{(26)}; \mathbf{h}_t^{(34)}], \\ \mathbf{c}_t &= \mathrm{RMSNorm}(W_c \mathbf{z}_t).\end{aligned} \tag{2}$$

The resulting context feature $\mathbf{c}_t$ is passed to every draft layer through its key and value (KV) cache. The same projection is used for every token and every draft layer [5].

DFlare instead learns a different linear combination of target states for each draft layer i [6]:

$$\begin{aligned}\boldsymbol{\alpha}^{(i)} &= \mathrm{softmax}(\mathbf{w}^{(i)}), \\ \mathbf{f}_t^{(i)} &= \mathrm{RMSNorm}\left(\sum_j \alpha_j^{(i)} \mathbf{h}_t^{(j)}\right).\end{aligned} \tag{3}$$

The weights depend on the draft layer i, but are fixed across token positions t once the model is trained. Our experiment instead keeps DFlash's projection and changes the weights based on the current hidden state.

Our modification keeps the same projection as DFlash but adds a linear gate before it. The gate takes the hidden state from layer 34 and computes five weights, one for each of the selected target layers. Each hidden state is multiplied by its corresponding weight before concatenation and projection.

We write the gate and the resulting projection as

$$\begin{aligned}\mathbf{g}_t &= G_\theta(\mathbf{h}_t^{(34)}), \\ \widetilde{\mathbf{z}}_t &= [g_{t,1}\mathbf{h}_t^{(2)}; \ldots; g_{t,5}\mathbf{h}_t^{(34)}], \\ \widetilde{\mathbf{c}}_t &= \mathrm{RMSNorm}(W_c \widetilde{\mathbf{z}}_t).\end{aligned} \tag{4}$$

G_θ is the learned gate and $g_{t,j}$ is the weight assigned to the j-th selected target layer at position t. The projection W_c is still learned in the same way as DFlash, only its input magnitude is adjusted. The gate is initialized so that $\mathbf{g}_t = \mathbf{1}$, making Equation 4 identical to Equation 2 at the start.

Unlike the fixed projection in DFlash, these weights are computed separately for each token, allowing the contribution of each target layer to depend on the context.

The gate introduces only 20,480 additional parameters.

4 Evaluation

Training. We train both models using a frozen Qwen3-8B target and the same 792,000 training examples in the same order. Both use DFlash's original training methodology with a block size of 16.

Training runs for one epoch, consisting of 24,750 updates with a global batch size of 32 on two H100 GPUs. We use 990 warmup steps, a peak learning rate of 6×10^{-4}, and cosine learning rate decay. We evaluate the final checkpoints after checking their training histories.

Math evaluation. After observing stronger performance on math prompts, we conduct a separate test using 700 held-out prompts from MetaMathQA and Orca-Math [7, 8]. Of these, 647 come from MetaMathQA and 53 from Orca-Math.

For each prompt, we provide the question without its answer. The target model generates the first token, and the draft model predicts the next 15 tokens. We then measure how many tokens match the target model's greedy predictions, including the first token generated by the target. We count acceptances from the beginning of the draft and stop at the first mismatch.

We use mean accepted length as our main metric to measure how closely the draft model predict the target's most likely token.

For our block size of 16, we count the first target token plus the number of draft tokens accepted before the first mismatch:

$$A = 1 + \max\{k : \hat{y}_j = y_j^* \text{ for all } 1 \le j \le k\}, \tag{5}$$

where $0 \le k \le 15$, $\hat{y}_j$ is a draft token, and y_j^* is the target's greedy token at that position. This definition matches the acceptance length reported in our evaluation.

Serving. We also measure decoding throughput using SGLang 0.5.18 on one A100 GPU. We select 24 fixed MetaMathQA prompts and generate 64 tokens per request using greedy decoding.

We test both models with eight concurrent requests. Each experiment is repeated three times using new servers, alternating the order in which the models are tested. Throughput is measured in output tokens per second.

5 Results

Table 1: Mean accepted tokens on 700 math prompts.

Model	Accepted tokens	Change
DFlash	2.98	–
DynaDFlash	**3.13**	**+5.0%**

On the 700 math prompts, the dynamic model achieves a mean accepted length of 3.13 tokens, compared to 2.98 for the original DFlash model. This is an improvement of about 5%, or 0.15 additional tokens per prompt.

The 95% confidence interval for the difference is 0.07 to 0.22 tokens, with $p < 0.001$.

Most of this improvement comes from the 647 MetaMathQA prompts, where the dynamic model accepts an additional 0.16 tokens on average. We do not observe an improvement on the smaller set of 53 Orca-Math prompts.

The results are different on our full evaluation, where the original DFlash model and the dynamic model have no statistically

significant difference in acceptance length. This suggests that the benefits of dynamic weighting may depend on the type of input rather than improving drafting across all tasks.

We also observe a throughput improvement when serving eight requests in parallel. The dynamic model generates an average of 1004.5 output tokens per second compared to 965.0 for DFlash, an improvement of 4.1%.

Table 2: Throughput is measured in output tokens per second.

Model	Output tokens/s	Change
Autoregressive Qwen3-8B	655.4	–
DFlash	965.0	–
DynaDFlash	1004.5	+4.1%

6 Discussion

Our results show that dynamic context projection can improve DFlash's acceptance length on math prompts, but the improvement does not extend to all types of prefixes. In particular, the dynamic model performs better on MetaMathQA while not significantly differing in performance from the baseline for the full evaluation.

We also looked at the weights produced by the gate to understand whether certain target layers were more useful for math. On the 700 math prompts, layers 2 had the largest mean gate, and layer 26 had the second largest. However, due to differences in the magnitude of the hidden features for these layers, layer 26 contributed the most to the projection, followed layer 34.

A key limitation is that both models were trained independently and do not match the acceptance length of the published DFlash model by z-lab. This makes the observed improvement in acceptance length less reliable as both models fall below the acceptance length of other DFlash drafters.

7 Conclusion

We tested whether dynamically weighting target hidden states before projection could improve DFlash's draft model. Our modified model achieves a 5% higher mean accepted length on 700 math prompts and a 4.1% throughput improvement in our serving test.

However, these improvements are not consistent across datasets, and we could not establish a clear relationship between dynamic context to the drafter and model performance. Further experimentation may be required with different methods to inject dynamic information from the target model to the drafter to demonstrate a gain across various contexts.

Acknowledgements

We thank Modal for supporting our work through a $5,000 research grant. All model training and evaluation was conducted on Modal's platform.

References

[1] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. *ICML*, 2023.

[2] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv:2302.01318*, 2023.

[3] Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *NeurIPS*, 2025.

[4] X. Cheng et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv:2607.05147*, 2026.

[5] J. Chen, Y. Liang, and Z. Liu. DFlash: Block diffusion for flash speculative decoding. *ICML*, 2026.

[6] J. Zhang et al. DFlare: Scaling up draft capacity for block diffusion speculative decoding. *arXiv:2606.02091*, 2026.

[7] L. Yu et al. MetaMath: Bootstrap your own mathematical questions for large language models. *ICLR*, 2024.

[8] A. Mitra et al. Orca-Math: Unlocking the potential of SLMs in grade school math. *arXiv:2402.14830*, 2024.

A Reproducibility and ablation

The training and evaluation code is in the DynaDFlash repository. The trained DFlash baseline and DynaDFlash models are on Hugging Face.

We trained a compact variant that sums the five gated target states before projection. This also makes the projection smaller. On the 700 math prompts, it accepts 2.96 tokens on average, compared with 2.98 for the DFlash baseline and 3.13 for DynaDFlash.