Accelerating Diffusion Transformer-Based Text-to-Speech with Transformer Layer Caching

Sakpiboonchit Siratish

Abstract. This paper presents a method to accelerate the inference process of diffusion transformer (DiT)-based text-to-speech (TTS) models by applying a selective caching mechanism to transformer layers. Specifically, I integrate SmoothCache into the F5-TTS architecture, focusing on caching outputs of self-attention and feed-forward network layers to reduce redundant computations during the denoising process. A calibration phase is introduced to analyze L1 relative errors between timesteps, guiding the selection of cache schedules that minimize quality degradation. To address the problem of inter-layer dependency, a unified caching schedule is adopted, applying the cache pattern derived from self-attention layers to both layer types. Experiments on LibriSpeech-PC and Seed-TTS datasets evaluate various cache thresholds and denoising step configurations. Results show that caching at higher denoising steps reduces inference time without compromising output quality, whereas caching at lower steps can negatively impact synthesis quality similarly to reducing the total number of denoising steps. Objective and subjective metrics confirm the effectiveness of SmoothCache in maintaining performance while improving computational efficiency. Comparisons between cached inference and reduced-step inference further highlight the benefits of selective caching, especially under high-step configurations. This work demonstrates that transformer layer caching is a practical solution for optimizing diffusion transformer-based TTS models without requiring architectural changes or retraining. Example inference results can be heard at `https://siratish.github.io/F5-TTS_SmoothCache/`.

1 Introduction

Text-to-Speech (TTS) systems have recently achieved remarkable progress [1,2], particularly with the rise of diffusion-based models capable of generating high-fidelity, natural-sounding speech conditioned on just a few seconds of prompt audio [3–5].

TTS models can be broadly categorized into autoregressive (AR) and non-autoregressive (NAR) frameworks. AR models, such as Tacotron 2 [6], generate speech sequentially, conditioning each frame on previous outputs. While this allows strong modeling of temporal dependencies, it results in slow inference and error accumulation during synthesis [7–9]. In contrast, NAR models, such as FastPitch [10], produce all frames in parallel, enabling faster synthesis without sequential dependencies. However, this parallelism introduces challenges in modeling long-range dependencies and necessitates explicit alignment mechanisms to ensure accurate mapping between input text and output speech [4,5,11].

Diffusion-based TTS models have emerged as an alternative that combines NAR efficiency with high synthesis quality [3, 5]. These models iteratively denoise latent audio representations, gradually converting noise into natural speech. While they avoid the alignment complexity of traditional NAR models, they remain computationally intensive due to repeated denoising steps across many timesteps.

Early examples such as E3-TTS [12] attempted to eliminate explicit phoneme-level alignment using cross-attention over character inputs but achieved limited audio quality. DiTTo-TTS used Diffusion Transformer (DiT) [13] with cross-attention conditioned on encoded text and improved synthesis fidelity by using pretrained language model to fine-tune the neural audio codec. E2-TTS [14] (based on Voicebox [4]) further simplified the pipeline by directly using padded character sequences as input and removing phoneme and duration predictors, although alignment robustness remained a concern.

Building upon these, F5-TTS [15] offers a streamlined yet effective approach. It discards explicit alignment modules and phoneme modeling by directly embedding padded character sequences via ConvNeXt V2 [16] and conditioning a DiT model with Flow Matching [17]. Its "Sway Sampling" inference strategy dynamically adjusts the sampling

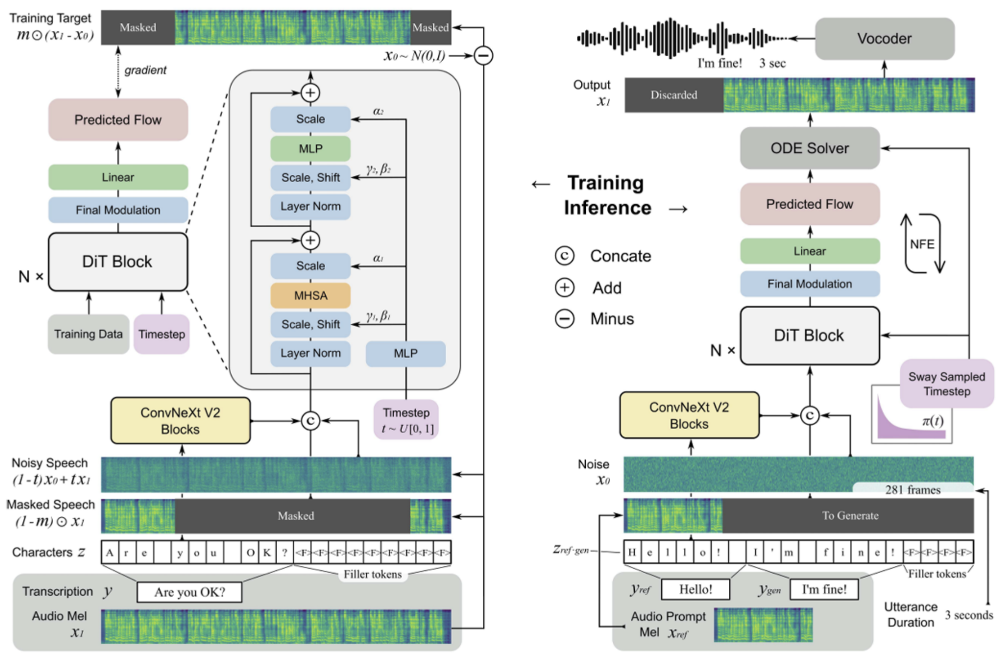

Fig. 1. Overview of F5-TTS training (left) and inference (right). The model is trained via text-guided speech infilling with Flow Matching loss. Input text is converted to padded character sequences and refined via ConvNeXt V2 before concatenation with speech input. Inference employs Sway Sampling to generate speech from noise.

process for faster generation. F5-TTS achieves a real-time factor (RTF) of approximately 0.15 while maintaining high naturalness, speaker similarity, and intelligibility. An overview of the F5-TTS architecture is illustrated in Fig.1.

Despite these advancements, F5-TTS, like other DiT-based models, remains constrained by the computational burden of iterative denoising. Prior work has explored various optimization strategies, including advanced solvers [18, 19], knowledge distillation [20], pruning [21], and quantization [22, 23].

A particularly promising zero-training, inference-only method is SmoothCache [24], an optimization technique designed to accelerate inference of DiT models. It exploits the observation that hidden representations in transformer layers exhibit redundancy across consecutive denoising timesteps. A short calibration pass measures cosine similarities to determine which layers and timesteps are safe to cache. These cached outputs are reused during inference, reducing computation without altering model parameters (see Fig.2 for schematic). While SmoothCache has demonstrated effectiveness in accelerating diffusion models for images, video, and audio generation, it has not yet been applied to TTS tasks.

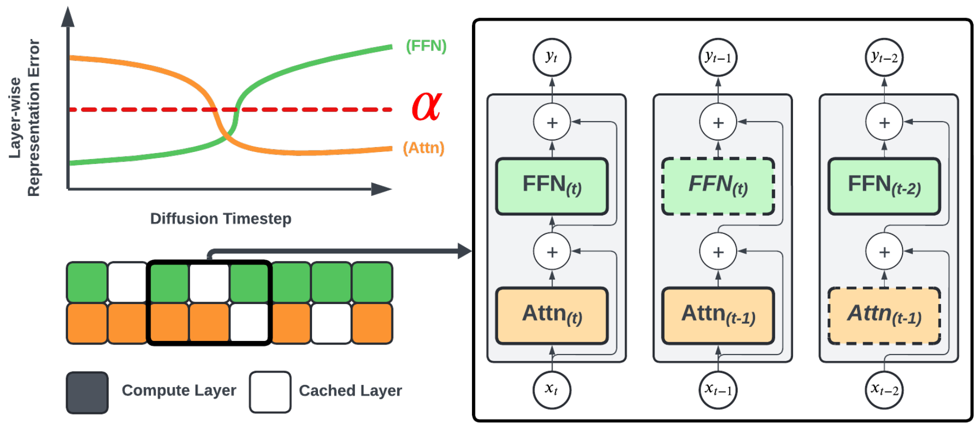

Fig. 2. Illustration of SmoothCache. During a calibration pass, layer-wise representation errors are analyzed. If the error falls below a threshold α, the layer output is cached for reuse in future timesteps. The left side shows the representation error trend; the right side depicts SmoothCache applied to a DiT model using residual connections to reintroduce cached outputs.

In this project, I investigate integrating SmoothCache into the F5-TTS pipeline without retraining the model. I evaluate caching strategies and assess their impact on inference speed and synthesis quality. Experimental results demonstrate that SmoothCache enables substantial inference acceleration with minimal degradation in output quality.

2 Data

This project uses the same evaluation datasets as the F5-TTS paper: 1) LibriSpeech-PC *test-clean* [25], and 2) Seed-TTS *test-en* [26].

2.1 LibriSpeech-PC *test-clean*

A filtered subset of LibriSpeech test-clean ($\sim$1000h, 16 kHz read English speech) focused on 4-10 second utterances. This subset contains 1,127 audio-text pairs ($\sim$2h total), drawn from 39 speakers. Audio is stored in FLAC format, named as `speakerID-chapterID-utteranceID.flac`. Corresponding transcripts are stored in per-chapter `.txt` files, where each line follows the format:

```
speakerID-chapterID-utteranceID TRANSCRIPTION_TEXT
```

The F5-TTS repository[1] also provides a `.lst` metadata file, in which each row consists of six tab-separated fields:

```
ref_utt ref_dur ref_txt gen_utt gen_dur gen_txt
```

Each field is defined as follows:

- **ref_utt**: Identifier of the reference utterance (e.g. `1234-5678-0001`)
- **ref_dur**: Duration of the reference audio in seconds
- **ref_txt**: Text transcript of the reference utterance
- **gen_utt**: Identifier assigned to the generated utterance (e.g. `1234-5678-0002`)
- **gen_dur**: Duration of the generated audio in seconds
- **gen_txt**: Text transcript used to generate the synthesized speech

Reference and generated utterances are paired via a cross-sentence sampling strategy, enabling objective quality evaluations.

2.2 Seed-TTS *test-en*

This test set is derived from Common Voice (English) [27], consisting of 1,088 utterances. Audio is explicitly split into reference and generated files. Filenames follow:

- `<ref_utt>.wav` — Reference Audio
- `<ref_utt>-<gen_utt>.wav` — Generated Audio

Metadata is provided in a `.lst` file, with each row containing:

```
utt prompt_text prompt_wav gt_text gt_wav
```

Fields are defined as follows:

- **utt**: Unique identifier for the utterance pair (e.g. `common_voice_en_00001-common_voice_en_00002`)
- **prompt_text**: Text transcript of the reference prompt
- **prompt_wav**: File path to the reference audio (`.wav`)
- **gt_text**: Text transcript intended for generation
- **gt_wav**: File path to the generated audio (`.wav`), if available

This metadata structure supports text-content alignment and separate access to reference vs. synthetic audio for evaluation.

[1] https://github.com/SWivid/F5-TTS

3 Methods and Algorithms

In this project I integrate SmoothCache into the F5-TTS inference pipeline in two stages: a Calibration Phase to discover a robust cache schedule, and an Inference Phase to apply that schedule and accelerate synthesis.

3.1 Calibration Phase

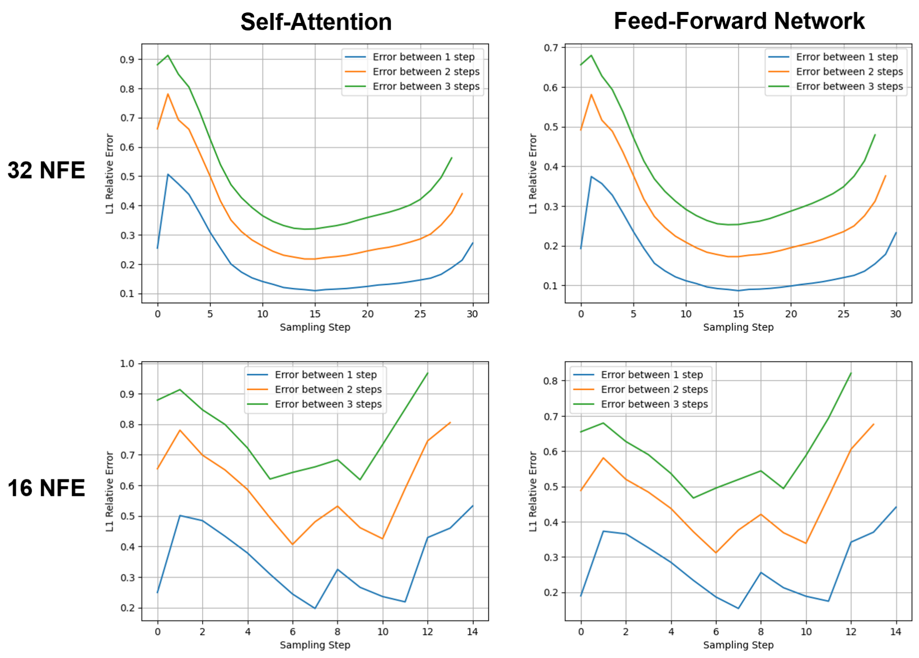

Fig. 3. L1 relative error between consecutive timesteps for self-attention and FFN layers, averaged over 10 samples. Curves are scaled to the same y-axis range.

In the Calibration Phase, I focus on the two most computationally intensive sub-layers of the Diffusion Transformer backbone: the self-attention (Attn) block and the feed-forward network (FFN) block, each followed by a residual connection. I process a few audio samples through the standard F5-TTS denoising loop and record the L1 relative error between consecutive timesteps for each layer. The resulting error curves, shown in Fig.3, highlight three trends: large errors at the beginning and end of the diffusion trajectory, consistently higher errors in Attn than in FFN, and an elevated baseline error when the number of steps is reduced. Using these insights, I select the default caching threshold α and generate an initial cache schedule for Attn and FFN.

However, the original cache schedules, derived by evaluating each layer in isolation, suffer from a key limitation noted in the SmoothCache paper: they do not account for interactions between layers when both are cached simultaneously [24]. In practice, applying these per-layer schedules caused noticeable audio artifacts, as caching one layer without recomputing the other disrupted the residual connections and degraded synthesis quality. To diagnose this, I conducted listening tests on a single audio example using five different caching strategies as shown in Fig.4: the original independent schedules, caching only Attn, caching only FFN, and two unified schedules that applies the pattern of one layer to both layers (Attn base and FFN base). These tests showed that "Attn-Only" introduced fewer artifacts than "FFN-Only", and that the unified (Attn base) schedule preserved Attn-Only audio fidelity while maximizing cache utilization. Consequently, I adopted this unified schedule for all subsequent experiments.

3.2 Inference Phase

In the Inference Phase, given a reference audio prompt (mel-spectrogram x_{ref}) and its transcription y_{ref}, along with a target text prompt y_{gen}, I inject the unified (Attn base) cache schedule into the F5-TTS denoising loop. At each diffusion step t, the model first denoises via an ODE solver with Sway Sampling. Before each residual connection in the Diffusion Transformer, I check whether t is marked for caching: if so, the corresponding layer output (Attn or FFN)

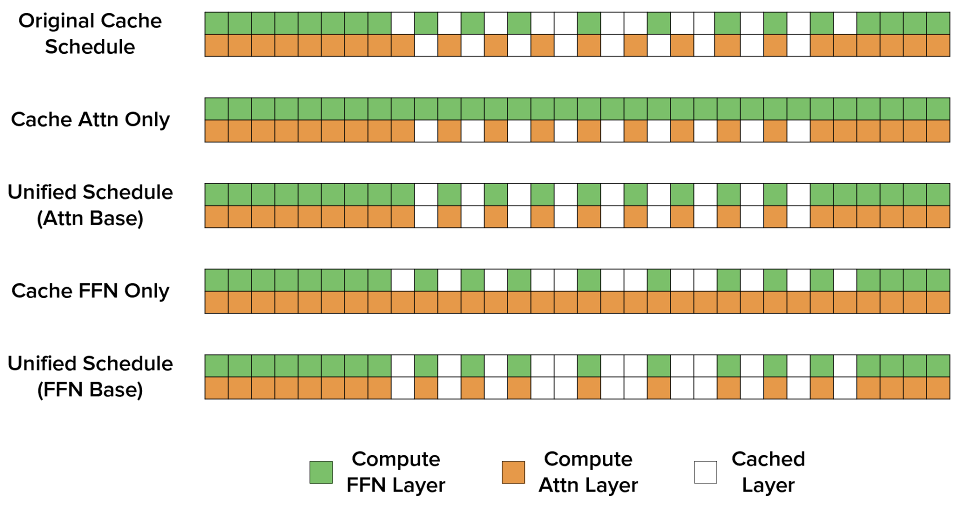

Fig. 4. Different caching strategies explored during calibration. The original cache schedule was derived using a threshold of $\alpha = 0.15$. Example inference results for each approach are available on the demo page.[2]

is retrieved from the cache; otherwise, it is computed normally and the cache is updated. After completing T steps, the final mel-spectrogram x_{gen} is converted into a waveform using the vocoder.

3.3 Experimental Setup

F5-TTS Setup I use the publicly available F5-TTS v1 base model[3] with exponential moving averaged (EMA) [28] weights. Inference is performed with the Euler ODE solver, a CFG [29] strength of 2, and a Sway Sampling coefficient of -1. The final log mel spectrograms are converted to waveform via the pretrained vocoder Vocos [30]. All hyperparameters and solvers match those in the original F5-TTS paper to ensure a controlled baseline.

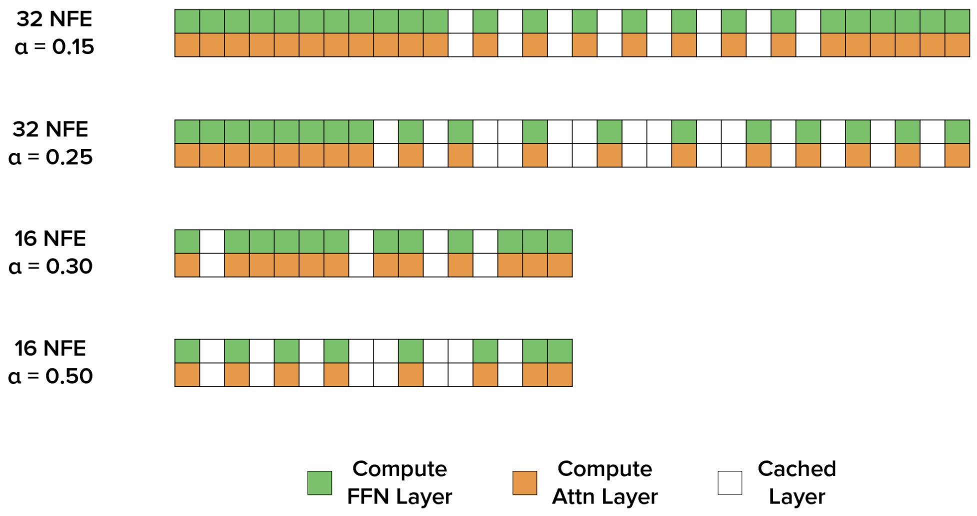

Fig. 5. Cache schedules with different NFE steps and caching thresholds (α) used for experiments. All schedules follow the Unified (Attn base) approach.

SmoothCache Setup For calibration, I randomly sample ten utterances from LibriSpeech-PC *test-clean*. I constrain the cache schedule so that no more than three consecutive timesteps are cached per layer, preventing excessive approx-

[2] https://siratish.github.io/F5-TTS_SmoothCache/
[3] https://huggingface.co/SWivid/F5-TTS

imation error. Cache schedules for all experiments are as shown in Fig.5. For 32 NFE, I compare the baseline to two thresholds: $\alpha = 0.15$, which results in caching approximately one-quarter of all steps, and $\alpha = 0.25$, which caches around half of the steps. For 16 NFE, I experiment with $\alpha = 0.30$ and $\alpha = 0.50$ to obtain cache schedules covering comparable proportions of timesteps. Those schedules are applied across the entire LibriSpeech-PC *test-clean* and Seed-TTS *test-en* datasets for evaluation.

Evaluation Metrics Objective evaluation includes Word Error Rate (WER), measured by transcribing generated speech with Whisper-large-v3 [31], and speaker Similarity (SIM-o), computed as the cosine similarity between WavLM-large [32] speaker embeddings of synthesized and ground-truth audio. For subjective evaluation, I record predicted mean opinion scores using two open-source MOS predictors: UTMOS strong [33] and NISQA v2.0 (mos only) [34]. I also conduct a user study with 37 participants, each judging 30 randomly selected utterance pairs in A/B trials on speaker similarity and overall quality. More details of the user study on Sec.4.1.

All evaluations follow the cross-sentence protocol [4, 35] on LibriSpeech-PC *test-clean* and Seed-TTS *test-en*. I report the average scores across three random seed generation results for each experiment to account for stochastic variability.

	LibriSpeech-PC *test-clean*				**Seed-TTS *test-en***				
Schedule	**WER(%)↓**	**SIM-o↑**	**UTMOS↑**	**NISQA↑**	**WER(%)↓**	**SIM-o↑**	**UTMOS↑**	**NISQA↑**	**RTF↓**
32 NFE (No Cache)	2.03	0.67	3.86	4.23	1.59	0.68	3.67	3.99	0.46
32 NFE ($\alpha = 0.15$)	2.05	0.68	3.86	4.21	1.58	0.68	3.69	4.01	0.32
32 NFE ($\alpha = 0.25$)	2.06	0.68	3.87	4.19	1.52	0.68	3.71	4.01	0.26
16 NFE (No Cache)	2.05	0.68	3.90	4.14	1.54	0.69	3.74	3.97	0.23
16 NFE ($\alpha = 0.30$)	2.02	0.68	3.87	4.09	1.55	0.69	3.74	3.96	0.18
16 NFE ($\alpha = 0.50$)	2.17	0.67	3.79	3.97	1.56	0.68	3.72	3.91	0.12

Table 1. Comparison results of F5-TTS on LibriSpeech-PC *test-clean* and Seed-TTS *test-en*. The Real-Time Factor (RTF) is computed with the inference time of 10s speech on The NVIDIA T4 GPU.

4 Discussion

Table 1 illustrates that applying SmoothCache to F5-TTS yields substantial inference speedups with minimal quality loss. For high NFE settings (e.g., 32 steps), caching approximately one quarter of the layers ($\alpha = 0.15$) reduces the RTF from 0.46 to 0.32, and caching half of the layers ($\alpha = 0.25$) further lowers RTF to 0.26, nearly a 2× speedup. Even at lower NFE (16 steps), caching accelerates inference (RTF from 0.23→0.18 for $\alpha = 0.30$), though overly aggressive caching ($\alpha = 0.50$) can degrade performance by eliminating too many compute steps.

A closer examination of the two evaluation datasets reveals subtle yet informative differences. On LibriSpeech-PC *test-clean*, which comprises read audiobook speech with consistent pacing and clarity, all caching configurations maintain nearly identical WER and speaker similarity scores. As the proportion of cached steps increases, subjective scores decline slightly but predictably, remaining within a narrow range. In contrast, Seed-TTS *test-en* encompasses a broader variety of speaking styles, speeds, and recording conditions, resulting in some inconsistency in perceptual metrics. For instance, NISQA scores on Seed-TTS improve marginally (3.99→4.01) at $\alpha = 0.15$ and 0.25, likely because the neural codec's resilience to diverse inputs masks minor caching artifacts; however, larger thresholds (e.g., $\alpha = 0.50$) produce the expected slight quality dips (3.97→3.91). These observations underscore that while SmoothCache delivers consistent speed-quality trade-offs on uniform datasets, it is less straightforward for more varied corpora.

Schedule	**WER(%)↓**	**SIM-o↑**	**UTMOS↑**	**NISQA↑**	**RTF↓**	**User Study**
32 NFE ($\alpha = 0.15$)	1.58	0.68	3.69	4.01	0.32	53%
24 NFE (No Cache)	1.54	0.68	3.68	3.99	0.31	47%
16 NFE ($\alpha = 0.30$)	1.55	0.69	3.74	3.96	0.18	49%
12 NFE (No Cache)	1.59	0.68	3.72	3.96	0.16	51%

Table 2. Comparison results of F5-TTS on Seed-TTS *test-en* under two strategies: (1) applying SmoothCache while maintaining the original number of NFE steps, and (2) reducing the number of NFE without caching, matched to the same number of compute steps. Includes results from a user study evaluating both settings.

4.1 Ablation of Caching Steps

Having established that SmoothCache can effectively optimize F5-TTS inference time without significant degradation in quality, I further investigate whether caching steps genuinely preserve synthesis performance compared to simply skipping those steps entirely. Specifically, according to Fig.5, since 32 NFE with caching threshold $\alpha = 0.15$ results in 24 compute steps (due to 8 cached steps), it is natural to compare this setup with a 24 NFE baseline without caching. Similarly, 16 NFE with $\alpha = 0.30$ yields 12 compute steps, motivating its comparison against a 12 NFE baseline without caching.

Table 2 summarizes the evaluation results from comparing these paired configurations. Across both pairs, all objective and subjective metrics appear closely matched. However, the user study results provide additional subjective insight: in the 32 NFE comparison, the cached version slightly outperforms the 24 NFE no-cache variant (53% vs. 47% preference), suggesting a marginal perceptual benefit from using cached steps rather than eliminating them outright. In contrast, the 16 NFE pair shows almost equal performance, indicating little distinction between caching and step reduction in this lower NFE setting.

These findings suggest that applying caching at higher NFE (longer generation trajectories) is more advantageous than reducing NFE steps directly. The reason may be that higher NFE schedules inherently possess more redundant or less impactful steps suitable for caching without affecting quality. In contrast, lower NFE schedules, which are already condensed, have fewer unimportant steps available for caching, meaning that caching at this level can harm synthesis quality similarly to reducing NFE steps. This analysis reinforces that SmoothCache is most beneficial when applied to higher NFE schedules, where it offers runtime acceleration without compromising output quality.

4.2 Limitations and Future Work

The primary limitation of this work lies in the restricted scope of experiments due to computational cost and time constraints. While the results demonstrate that SmoothCache effectively reduces inference time without significantly degrading output quality, there remain several unexplored avenues that could lead to further improvements.

In the calibration process, although applying the cache schedule derived from the Attn layer to both Attn and FFN layers successfully mitigated dependency issues, this approach is admittedly heuristic. A more principled strategy could involve integrating L1 relative error curves from both layers when constructing cache schedules, potentially weighting the contribution from each layer or treating them equally. Additionally, using alternative datasets or different sampling strategies for calibration could reveal whether the observed error distributions are model- or dataset-specific.

Regarding the inference process, the current experiments are limited to varying the number of denoising steps (NFE) and caching thresholds. Future work should investigate how SmoothCache interacts with other inference hyperparameters, such as CFG strength, Sway Sampling coefficient, and different ODE solvers (e.g., midpoint or Heun-3). Similarly, exploring its performance with different vocoders, such as BigVGAN [36], could provide broader insights into its generalizability across synthesis pipelines.

In terms of evaluation, the current subjective assessment is limited to direct A/B preference tests comparing output pairs. Expanding user studies to include tests focused on specific perceptual attributes could yield richer insights. For instance, conducting CMOS tests (comparing each generated sample against a reference and rating from -3 to +3) would provide quantitative data on overall audio quality relative to natural speech. Similarly, SMOS tests (speaker similarity ratings from 1 to 5) could offer a more focused measure of speaker identity preservation.

5 Conclusion

This study investigated the application of SmoothCache to accelerate F5-TTS inference by selectively caching transformer layer outputs without modifying or retraining the base model. Through a calibration process based on L1 relative error analysis and a unified caching strategy to address inter-layer dependencies, SmoothCache achieved significant reductions in inference time while maintaining synthesis quality. Experiments on LibriSpeech-PC and Seed-TTS demonstrated that caching at higher NFE steps preserves quality more effectively than reducing denoising steps, whereas caching at lower NFE levels degrades performance comparably to step reduction. The results suggest that calibrated caching offers a practical approach to optimizing diffusion-based TTS models. Future work could enhance calibration strategies, explore more varied inference conditions, and refine evaluation methodologies to extend the applicability and effectiveness of SmoothCache.

References

[1] Wang T, Zhou L, Zhang Z, Wu Y, Liu S, Gaur Y, et al. VioLA: Unified Codec Language Models for Speech Recognition, Synthesis, and Translation. arXiv. 2023.

[2] Tan X, Chen J, Liu H, Cong J, Zhang C, Liu Y, et al. NaturalSpeech: End-to-End Text-to-Speech Synthesis With Human-Level Quality. IEEE transactions on pattern analysis and machine intelligence. 2024;46(6):4234-45.

[3] Shen K, Ju Z, Tan X, Liu Y, Leng Y, He L, et al. NaturalSpeech 2: Latent Diffusion Models are Natural and Zero-Shot Speech and Singing Synthesizers. arXiv. 2023.

[4] Le M, Vyas A, Shi B, Karrer B, Sari L, Moritz R, et al. Voicebox: Text-Guided Multilingual Universal Speech Generation at Scale. arXiv. 2023.

[5] Ju Z, Wang Y, Shen K, Tan X, Xin D, Yang D, et al. NaturalSpeech 3: Zero-Shot Speech Synthesis with Factorized Codec and Diffusion Models. arXiv. 2024.

[6] Shen J, Pang R, Weiss RJ, Schuster M, Jaitly N, Yang Z, et al. Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2018;00:4779-83.

[7] Han B, Zhou L, Liu S, Chen S, Meng L, Qian Y, et al. VALL-E R: Robust and Efficient Zero-Shot Text-to-Speech Synthesis via Monotonic Alignment. arXiv. 2024.

[8] Peng P, Huang PY, Li SW, Mohamed A, Harwath D. VoiceCraft: Zero-Shot Speech Editing and Text-to-Speech in the Wild. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024:12442-62.

[9] Song Y, Chen Z, Wang X, Ma Z, Chen X. ELLA-V: Stable Neural Codec Language Modeling with Alignment-Guided Sequence Reordering. Proceedings of the AAAI Conference on Artificial Intelligence. 2025;39(24):25174-82.

[10] Lancucki A. Fastpitch: Parallel Text-to-Speech with Pitch Prediction. ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2021;00:6588-92.

[11] Mehta S, Tu R, Beskow J, Székely Henter GE. Matcha-TTS: A Fast TTS Architecture with Conditional Flow Matching. ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2024;00:11341-5.

[12] Gao Y, Morioka N, Zhang Y, Chen N. E3 TTS: Easy End-to-End Diffusion-Based Text To Speech. 2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU). 2023;00:1-8.

[13] Peebles W, Xie S. Scalable Diffusion Models with Transformers. 2023 IEEE/CVF International Conference on Computer Vision (ICCV). 2023;00:4172-82.

[14] Eskimez SE, Wang X, Thakker M, Li C, Tsai CH, Xiao Z, et al. E2 TTS: Embarrassingly Easy Fully Non-Autoregressive Zero-Shot TTS. 2024 IEEE Spoken Language Technology Workshop (SLT). 2024;00:682-9.

[15] Chen Y, Niu Z, Ma Z, Deng K, Wang C, Zhao J, et al. F5-TTS: A Fairytaler that Fakes Fluent and Faithful Speech with Flow Matching. arXiv. 2024.

[16] Woo S, Debnath S, Hu R, Chen X, Liu Z, Kweon IS, et al. ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023;00:16133-42.

[17] Lipman Y, Chen RTQ, Ben-Hamu H, Nickel M, Le M. Flow Matching for Generative Modeling. arXiv. 2022.

[18] Liu X, Gong C, Liu Q. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. arXiv. 2022.

[19] Lu C, Zhou Y, Bao F, Chen J, Li C, Zhu J. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models. Machine Intelligence Research. 2025:1-22.

[20] Salimans T, Ho J. Progressive Distillation for Fast Sampling of Diffusion Models. arXiv. 2022.

[21] Fang G, Ma X, Wang X. Structural Pruning for Diffusion Models. arXiv. 2023.

[22] He Y, Liu L, Liu J, Wu W, Zhou H, Zhuang B. PTQD: Accurate Post-Training Quantization for Diffusion Models. arXiv. 2023.

[23] Shang Y, Yuan Z, Xie B, Wu B, Yan Y. Post-Training Quantization on Diffusion Models. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023;00:1972-81.

[24] Liu J, Geddes J, Guo Z, Jiang H, Nandwana MK. SmoothCache: A Universal Inference Acceleration Technique for Diffusion Transformers. arXiv. 2024.

[25] Meister A, Novikov M, Karpov N, Bakhturina E, Lavrukhin V, Ginsburg B. LibriSpeech-PC: Benchmark for Evaluation of Punctuation and Capitalization Capabilities of End-to-End ASR Models. 2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU). 2023;00:1-7.

[26] Anastassiou P, Chen J, Chen J, Chen Y, Chen Z, Chen Z, et al. Seed-TTS: A Family of High-Quality Versatile Speech Generation Models. arXiv. 2024.

[27] Ardila R, Branson M, Davis K, Henretty M, Kohler M, Meyer J, et al. Common Voice: A Massively-Multilingual Speech Corpus. arXiv. 2019.
[28] Karras T, Aittala M, Lehtinen J, Hellsten J, Aila T, Laine S. Analyzing and Improving the Training Dynamics of Diffusion Models. 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2024;00:24174-84.
[29] Ho J, Salimans T. Classifier-Free Diffusion Guidance. arXiv. 2022.
[30] Siuzdak H. Vocos: Closing the gap between time-domain and Fourier-based neural vocoders for high-quality audio synthesis. arXiv. 2023.
[31] Radford A, Kim JW, Xu T, Brockman G, McLeavey C, Sutskever I. Robust Speech Recognition via Large-Scale Weak Supervision. arXiv. 2022.
[32] Chen Z, Chen S, Wu Y, Qian Y, Wang C, Liu S, et al. Large-Scale Self-Supervised Speech Representation Learning for Automatic Speaker Verification. ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2022;00:6147-51.
[33] Saeki T, Xin D, Nakata W, Koriyama T, Takamichi S, Saruwatari H. UTMOS: UTokyo-SaruLab System for VoiceMOS Challenge 2022. arXiv. 2022.
[34] Mittag G, Naderi B, Chehadi A, Möller S. NISQA: A Deep CNN-Self-Attention Model for Multidimensional Speech Quality Prediction with Crowdsourced Datasets. Interspeech 2021. 2021:2127-31.
[35] Chen S, Wang C, Wu Y, Zhang Z, Zhou L, Liu S, et al. Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers. IEEE Transactions on Audio, Speech and Language Processing. 2025;33:705-18.
[36] Lee Sg, Ping W, Ginsburg B, Catanzaro B, Yoon S. BigVGAN: A Universal Neural Vocoder with Large-Scale Training. arXiv. 2022.