REDIT: Resilient Distributed Text-to-Speech at Edge Networks

Teng Li, Hulya Seferoglu, and Erdem Koyuncu
tli81@uic.edu, hulya@uic.edu, ekoyuncu@uic.edu
University of Illinois at Chicago

***Abstract*—Existing deep learning-based Text-to-Speech (TTS) mechanisms are computationally intensive, which puts a strain in their practical applications especially over edge networks comprised of resource constrained devices. Our focus is on distributing TTS tasks across multiple devices (i.e., workers) at edge networks and providing TTS-aware resiliency against straggling workers. In particular, we design a REsilient DIstributed Tts (REDIT) framework by exploiting the text summarization as redundancy to provide resiliency for distributed TTS. We show analytically that REDIT improves the task completion time as compared to the distributed TTS without resiliency. We determine the optimum amount of redundancy/summary based on our task completion time analysis. We implement our REDIT framework in a real testbed consisting of NVIDIA Jetson Nano cards, and show that our REDIT algorithm improves the task completion delay as compared to baselines.**

I. Introduction

Text-to-speech (TTS), also known as speech synthesis, converts text into human speech. The recent developments in deep-learning have increased speech quality of TTS significantly, even matching natural human voice [1]–[3]. However, existing deep-learning-based TTS mechanisms are computationally intensive, which puts a strain in their practical applications especially over edge networks comprised of resource constrained devices. There are some approaches, e.g., [4]–[6] to reduce the time complexity of TTS by modifying learning models. In this paper, we follow a complementary approach of distributing TTS tasks across multiple devices (such as edge servers and end users) at edge networks.

Although distributed computing is a de facto approach to speed up computationally intensive applications, its potential diminishes when some workers (devices that tasks are offloaded) are delayed or fails (known as stragglers). Such stragglers become a bottleneck for the completion time of the distributed task. Stragglers problem, despite seen in cloud computing with dedicated servers, is emphasized more in edge computing systems where computing entities such as edge servers and end users have heterogeneous and time-varying resources.

One promising solution to address the stragglers problem is adding redundancy, where the replica of some tasks are assigned to multiple workers [7], [8]. This approach will reduce the probability of stragglers even if one worker is delayed or fails, another worker that processes the same task could finish the task on time. However, it is crucial to optimize the amount of redundancy as too much redundancy will result in under utilization of precious system resources. Coded distributed computation is a recent area focusing on the usage of error correcting codes to optimize the amount of redundancy [9]. This area, although very promising, falls short of dealing with non-linear computations, which is inherent in TTS. In this paper, we consider a TTS-aware redundancy adding mechanism to address straggling workers in distributed TTS.

It is a fact that human language has some inherent redundancy, which means that it is usually possible to express the key information in a much shorter form. Our goal is to use this redundancy in our resilient and distributed TTS. In particular, we employ text summarization [10] to reduce the redundancy in the text that is supposed to be converted to speech. The summarization is used as "joker" and converted to speech when the TTS of the original text fails due to straggling workers.

In this paper, we design a REsilient DIstributed Tts (REDIT) framework by exploiting the text summarization as redundancy. A master device splits a text into smaller texts (sub-tasks), and offloads them to workers for TTS, where workers perform TTS. Meanwhile, the master device performs text summarization. The master device performs TTS of the summarized text. If there are straggling workers, which delays TTS, the master

This work was supported in parts by the Army Research Lab (ARL) under Grant W911NF-2120272 and National Science Foundation (NSF) under Grants CCF-1942878 and CNS-2112471.

device can use the TTS of the summary. We show analytically that REDIT improves the task completion time as compared to the distributed TTS which does not have resiliency. We determine the optimum amount of redundancy/summary based on our task completion time analysis. We implement our REDIT framework in a real test bed consisting of NVIDIA Jetson Nano cards [11].

The structure of the rest of this paper is as follows. Section II presents the related work. Section III presents our system model. Section IV presents our resilient and distributed TTS (REDIT) framework with task completion time analysis and the redundancy/summary optimization. Section V provides experimental evaluation of REDIT in real devices. Section VI concludes the paper.

II. Related Work

The state-of-the-art TTS methods, especially deep-learning-based methods, are computationally intensive. For example, [12] provides a performance test on TTS frameworks Tacontron2 and WaveGlow, and shows that their inference tasks introduce heavy workloads for high-performance GPUs like NVIDIA T4. It becomes very inefficient to run such large-scale TTS algorithms on edge devices such as NVIDIA Jetson Nano or smartphones. Task distribution is a straightforward idea to improve the speed of TTS over edge networks.

There are some distributed Natural Language Processing (NLP) mechanisms in the literature. A distributed learning algorithm to speed up the training of NLP models, which is based on asynchronous mini-batch learning is designed in [13]. However, it focuses on the training phase and ignores the distributed inference or deployment. TextImager [14] designs distributed NLP solutions, but it is not based on deep learning and did not have resiliency aspects. A distributed TTS framework on embedded devices is built in [15]. The basic idea is to divide a TTS pipeline into the front-end and the back-end, and send them to server and clients separately.

Straggling workers is an important challenge for distributed systems [7]. This problem is usually addressed with replication-based redundancy such as replicating tasks over multiple workers [7], [8], which may potentially introduce too much redundancy and leads to under utilization of resources. In this paper, we consider a TTS-aware redundancy adding mechanism to address straggling workers in distributed TTS.

III. Model

Setup. We consider a distributed computing system formed of connected computing entities; end devices, edge servers, and cloud. We divide these computing entities into (i) masters who want to perform intensive TTS conversions; or (ii) workers who are willing to dedicate some of their resources to help in the computations. There could be multiple masters and workers in the system, which may overlap.

Master/Worker Model. We focus on a master/worker setup, where the master device offloads its computationally intensive TTS tasks to Worker $n \in \mathcal{N}$ (where $\mathcal{N} \triangleq \{1, \ldots, N\}$ is the set of all workers) via device-to-device (D2D) links such as Wi-Fi Direct.

The workers have the following properties: (i) Workers may fail or "sleep/die" or leave the network before finishing their assigned computational tasks. (ii) Workers incur delays in responding to the master, where delay has two components; (i) transmission delay for exchanging text and speech, and (ii) computation delay.

Text-to-Speech (TTS). The master device divides a long text message into smaller texts, and offloads them to workers. The workers convert text to speech using TTS model. Our work is compatible with any TTS model, but we focus on Tacotron [3] in this paper. Tacotron is a learning-based end-to-end generative model for TTS and can be applied to many languages, like English, Chinese Mandarin, Korean, etc. With "end-to-end", we mean that it can be trained by <text, audio> pairs, also with the input of texts and output of generated speeches, which makes it easier to train and use. Tacotron achieves very good voice quality [3]; it achieves a higher Mean Opinion Score (MOS), a common metric for evaluating speech quality, than its competitors. One of the disadvantages of Tacotran is that it is not distributed. Therefore, in this paper, we first focus on implementing Tacotron in a distributed manner, then we focus on resiliency.

Text Summarization. We use T5, which is a natural language processing (NLP) framework based on transfer learning, designed with the goal of dealing with all text-based language problems including text summarization, question answering, machine translation, etc. In this paper, we choose a small size T5 model, T5-small, as the summarizer. T5-small has around 60 million parameters and is a more suitable choice in our edge network setup. The T5-small used in our project is pre-trained using C4 dataset [16], [17]. We note that REDIT is able to work with other summary models, including learning- and non-learning-based, like BERT [18], [19] and BERT-based improvements [20]–[22].

IV. REDIT: Resilient and Distributed TTS

Design of REDIT. We develop a REDIT algorithm, where the master device pre-processes the original text

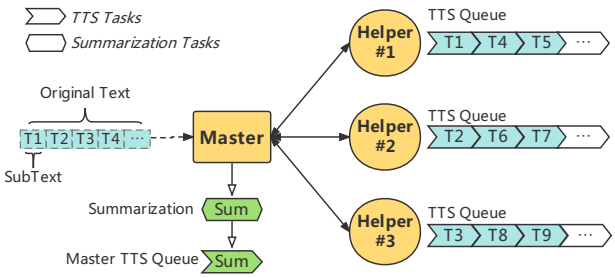


Fig. 1: Resilient Distributed TTS. The master device is responsible for TTS of the summary.

Fig. 2: Experiment setup.

and splits it into multiple subtexts. The created subtexts are kept in a waitlist in the master device.

The master device and workers always have an open connection, i.e., the master device detects when a worker is idle. The subtexts are transmitted from the waitlist in the master device one by one to idle workers. In other words, when a worker is idle, it requests a task from the master, and a new subtext is taken from the waitlist and transmitted to that worker. This continues until all the subtexts are offloaded to workers.

Meanwhile, the master device sets up another thread for text summarization and TTS of the original text. The master device runs two threads in parallel; (i) text splitting and offloading to workers, and (ii) text summarization and TTS of the summarized text. When offloading of subtexts is finished, the master device waits for receiving either (i) the audios corresponding to all offloaded subtexts, or (ii) the audio of the summarized text. If one of these events is realized, then TTS tasks end. If the audios of the subtexts are received, the master device stitches them to one single audio.

In REDIT, even if the master device cannot get the audio of the original text, it can still get the summary. The audio of the summary still carries important information thanks to redundancy in human speech.

Analysis of REDIT. Assume that Worker n processes all of its subtexts in Y_n time duration, which follows exponential distribution with rate λ, which is a common assumption in distributed computation with stragglers [9]. The time it takes for text summarization and converting the summary to speech at the master device takes $X(s)$ time duration following an exponential distribution with rate $\mu(s)$, where s is the summary rate. The summary rate $0 \leq s \leq 1$ increases with the summary size, and becomes 1 when the summary size equals to the size of the original text.

Theorem 1: The task completion time of REDIT is Z, and its expected value is

$$\mathrm{E}[Z] = \sum_{i=1}^{N} \frac{(-1)^{i-1}}{\mu(s)+\lambda i} \binom{N}{i}. \tag{1}$$

For the special case $\mu(s) = \lambda$, we have $\mathrm{E}[Z] = \frac{1}{\lambda}\frac{N}{N+1}$.

Proof: Let $\overline{Y} \triangleq \max_i Y_i$, $Z \triangleq \min\{X(s), \overline{Y}\}$. We have

$$\begin{aligned} \mathrm{P}(Z \leq z) &= 1 - P(X(s) \geq z)P(\overline{Y} \geq z) && (2)\\ &= 1 - P(X(s) \geq z)(1 - \textstyle\prod_{i=1}^{N} P(Y_i \leq z)) && (3)\\ &= 1 - e^{-\mu(s)z}(1-(1-e^{-\lambda z})^N), && (4) \end{aligned}$$

so that

$$\begin{aligned} f_Z(z) = \frac{\mathrm{dP}(Z \leq z)}{\mathrm{d}z} &= \mu(s)e^{-\mu(s)z}(1-(1-e^{-\lambda z})^N) + \\ & N\lambda e^{-(\lambda+\mu(s))z}(1-e^{-\lambda z})^{N-1}. \end{aligned} \tag{5}$$

We can now calculate the expected value of Z as $\mathrm{E}[Z] = \int_0^\infty z f_Z(z)\mathrm{d}z$. To evaluate the integral, we first expand powers of $(1-e^{-\lambda z})$ that appear in (5) via the binomial theorem, and use the fact that $\int_0^\infty z e^{-\alpha z} = \frac{1}{\alpha^2}$, $\alpha > 0$ for different values of α. After these cumbersome but straightforward calculation steps, we arrive at

$$\mathrm{E}[Z] = \sum_{i=1}^{N} \frac{(-1)^{i-1}}{\mu(s)+\lambda i} \binom{N}{i} \tag{6}$$

For the special case $\mu(s) = \lambda$, we have

$$\lambda \mathrm{E}[Z] = \sum_{i=1}^{N} \frac{(-1)^{i-1}}{1+i} \binom{N}{i} = \frac{N}{N+1}. \tag{7}$$

The last equality can be obtained after some straightforward algebraic manipulations and the binomial theorem. This concludes the proof. □

On the other hand, the task completion of the distributed TTS without text summarization is W, where $W = \overline{Y} \triangleq \max_i Y_i$. It is well-known (see e.g. [23, Eq. 7.10]) that $\lambda \mathrm{E}[W] = 1 + \frac{1}{2} + \cdots + \frac{1}{N} = \log N + \gamma + o(1)$,

where the second equality holds as $N \to \infty$, and $\gamma = 0.577\cdots$ is the Euler-Mascheroni constant. As seen, REDIT improves the task completion delay on the order of $\log N$ as compared to the distributed TTS scenario without text summarization, which is significant.

Optimization of REDIT. It is crucial to determine the summary rate s in REDIT framework. The quality of experience will be higher for large s values, because the summary will provide more information when s is large. On the other hand, this will increase the task completion time. We use our task completion time analysis in (1) to determine the summary rate when there is constraint on task completion time. The optimization problem is

$$\max\ s \quad \text{s.t.} \quad E[Z] = \sum_{i=1}^{N} \frac{(-1)^{i-1}}{\mu(s) + \lambda i} \binom{N}{i} \leq Z_{\text{thr}}, \quad (8)$$

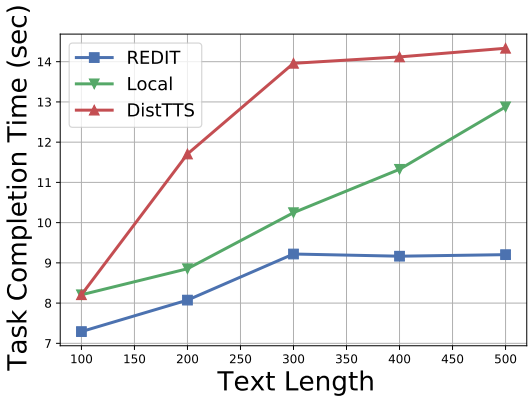
where Z_{thr} is a task completion delay constraint. It is straightforward to see that $E[Z]$ is a monotonically non-decreasing function of s, because $X(s)$ and $Z = \min\{X(s), \overline{Y}\}$ are monotonically non-decreasing functions of s. Thus, we can use binary search to solve (8).

V. Experimental Evaluation

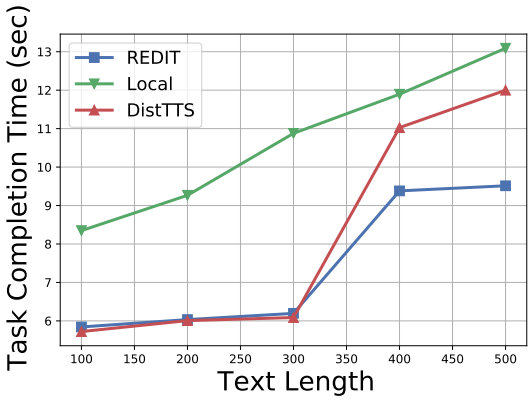
Setup: The experiment platform comprises multiple computation nodes equipped with different types of hardware. The master device is a high-end PC equipped with AMD Ryzen 3900x 12-cores CPU @3800Mhz, which is CPU-only; in other words, all services are processed by CPU in the master. The worker nodes are based on Jetson Nano Developer Kits, which are implemented with ARM A57 4-cores CPU@1479MHz and 128-core Maxwell GPU @921.6 Mhz. Different from the master device, the GPU will accelerate the services at the workers if the service supports GPUs. In our experiments, we have three workers.

Dataset. We use a pre-trained Tacotron model, which is trained by using the LJ speech dataset [24]. We assembled a dataset based on CNN and BBC news [25] as the experiment input. The dataset includes texts with 100 to 500 characters. We set the subtext size roughly to 100 characters as we observed that Tacotron gives the best performance when the input text size is 100 when it is trained by the LJ speech dataset. For example a 600 character text is divided into 6 subtexts, where each subtext is around 100 characters.

Delay Model. One of the workers is randomly selected and acts as a straggler, which performs additional computationally intensive tasks (performs TTS of a randomly generated text) to introduce delay.



(a) One worker is a straggler



(b) No straggler

Fig. 3: Task completion time versus text length when (a) one of the workers is a straggler, (b) none of the workers is a straggler.

Baselines. We consider two baselines; (i) Local: The master device performs the TTS of the original text without any task offloading. (ii) DistTTS: Distributed TTS similar to REDIT without text summarization.

Fixed Summary Size. Fig. 3(a) shows the average task completion time (averaged over 50 experiments) versus the text length. One of the workers is randomly selected as a straggler. We set the maximum length of the text summary to be 15 words for all text sizes. In this scenario, DistTTS performs worse than REDIT and Local as one of the workers is a straggler and creates a bottleneck for DistTTS. Local improves on DistTTS as it does not rely on any workers for TTS. In other words, the task completion time of Local is not affected by straggling workers. The completion time of Local increases linearly with increasing text length as there is no distribution in this setup, and there is a linear relationship with the time complexity of TTS with input text size. REDIT performs better than both Local and DistTTS thanks to using text summarization and its TTS. As seen, the task completion

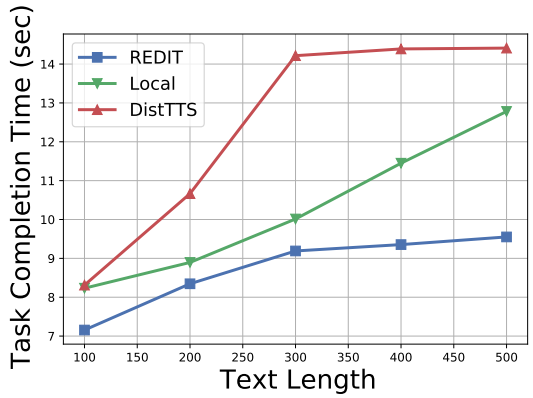


Fig. 4: Task completion time versus text length when the summary size is proportional to the text length.

time of REDIT increases linearly first, then stays the same. The reason is that it uses distributed TTS and text summarization alternatingly depending on the delay the straggling worker imposes when text length is smaller. However, it uses text summarization when text length increases as TTS of the original text becomes costly, so we see a flat curve for larger text lengths.

We consider the same setup in Fig. 3(b), but we consider an ideal scenario that workers do not straggle. Fig. 3(b) shows the average task completion time versus text length. Local performs worse than REDIT and DistTTS as workers are operating in their full performance without any delay. REDIT and DistTTS have the same performance when the text lengths are smaller, because TTS of the original text is the most efficient scenario in this case and REDIT and DistTTS are doing exactly the same operations; i.e., offloading subtexts, receiving corresponding audios, and stitching them. On the other hand, when the text lengths increase, text summarization and its TTS becomes more efficient, so REDIT resorts to this option and performs better than DistTTS. We see almost a flat curve for REDIT and DistTTS when text lengths are 100, 200, and 300. Noting that the size of sub-texts is 100, only one worker is used when text length is 100, two parallel workers are used when text length is 200, and three parallel workers are used when the text length is 300. In all these cases, the task completion time is the same as workers operate in parallel and in their full capacity. We see a jump when text length is 400 as three subtexts are processed first, and then another subtask is processed, which increases the task completion time.

Fig. 4 considers the same setup as in Fig. 3(a), except the summary size. We arrange the summary size proportional to the length of the actual summary. In particular, the ratio of summary size (words) to text length (characters) is 0.05. This means that the summary size is 25 words when the text length is 500 characters. The resulting task completion time versus text length is shown in Fig. 4. REDIT performs better than both DistTTS and Local. This is similar to Fig. 3(a), which shows that REDIT works with different summary sizes.

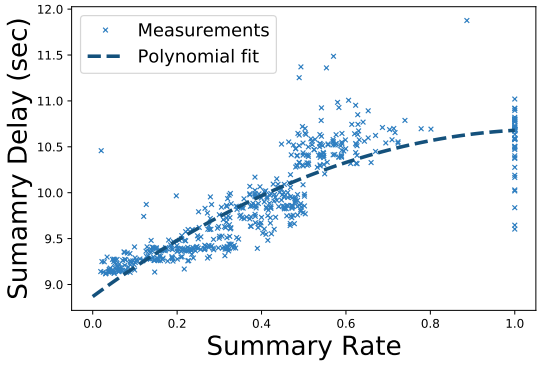


Fig. 5: Measurement of summary delay; i.e., $\frac{1}{\mu(s)}$ versus s. There is one straggler.

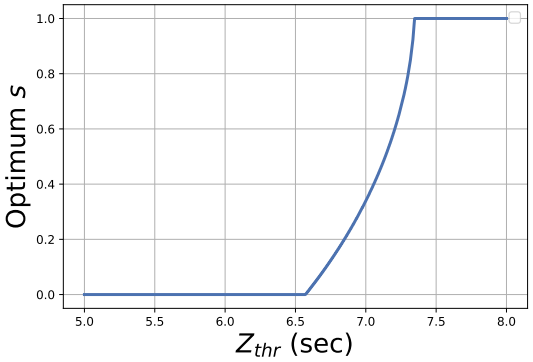


Fig. 6: Optimum summary size versus Z_{thr}. There is one straggler.

Optimum Summary Size. Now we determine the optimum summary rate as a solution to (8). In particular, We determine λ and $\mu(s)$ based on our experiments, and determine s by solving (8). We consider the same setup in Fig. 3 with one straggler. We measured $\lambda = 0.116$. We fitted $\frac{1}{\mu(s)}$ to a polynomial $a + bs + cs^2$ as seen in Fig. 5, where $a = 8.8682, b = 3.3701, c = -1.5587$. We can use this polynomial for the solution of (8).

The summary rate as a solution of (8) for different Z_{thr} is shown in Fig. 6. As seen, the optimum summary rate increases with increasing Z_{thr}. The solutions of the polynomial $8.8682 + 3.3701s - 1.5587s^2$ are 6.5702 and 7.3460, which is also observed in Fig. 6.

Mean Opinion Score (MOS) Experiments. We also conducted Mean Opinion Score (MOS) survey based on

TABLE I: MOS Scores of REDIT and Non-distributed Tacotron

	MOS with 95% Confidential Interval
REDIT	3.519±0.358
Non-distributed Tacotron	3.675±0.505

Absolute Category Rating (ACR) to evaluate the quality of speech synthesis of REDIT as compared to Tacotron. In our survey, we created ten groups of speech samples, where each group has different speech samples (each of them corresponds to the TTS of 200 character texts). There are three clips in each group (same speech sample), where two are generated by REDIT via 'stitching the portions together", while the other clip is generated by the pre-trained Tacotron without any splitting and stitching. The order of each speech generator within each group is shuffled randomly across different groups. Within each group, the responder is asked to evaluate the quality of speech for each sample from best (5) to worst (1), and is blind to any remaining information. Our MOS scores are shown in Table I.

The table shows that the MOS score of REDIT is very close to Tacotron. The slight performance reduction is due to stitching the portions together in REDIT and expected. The results shows the effectiveness of our stitching mechanism. Besides, REDIT is a general framework that can be suitable for other TTS models including learning-based and non-learning-based models. In our project, we used Tacotron as an example. Other models, such as Tacotron2, Transformer TTS, FastSpeech can also be used. Obviously, the synthesis quality highly depends on the chosen model. We believe that, by replacing the simple pre-trained Tacotron with other TTS methods, the MOS can be improved significantly.

VI. Conclusion

We design a REsilient DIstributed Tts (REDIT) framework by exploiting the text summarization as redundancy to provide resiliency for distributed TTS. In REDIT, the master device splits a text into smaller texts (sub-tasks), and offloads them to workers for TTS. Meanwhile, the master device performs text summarization as well as the TTS of the summarized text. If there are straggling workers, which delays TTS, the master device can use the TTS of the summary. We showed through analysis and implementation in a real testbed that REDIT improves the task completion delay as compared to baselines.

References

[1] Y. Ning, S. He, Z. Wu, C. Xing, and L.-J. Zhang, "A review of deep learning based speech synthesis," *Applied Sciences*, vol. 9, no. 19, p. 4050, 2019.

[2] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[3] Y. Wang *et al.*, "Tacotron: Towards end-to-end speech synthesis," *arXiv:1703.10135*, 2017.

[4] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6706–6713.

[5] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech: Fast, robust and controllable text to speech," *arXiv preprint arXiv:1905.09263*, 2019.

[6] Y. Ren, C. Hu, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech 2: Fast and high-quality end-to-end text to speech," *arXiv preprint arXiv:2006.04558*, 2020.

[7] M. Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems," *arXiv preprint cs/0501002*, 2005.

[8] A. Ledmi, H. Bendjenna, and S. M. Hemam, "Fault tolerance in distributed systems: A survey," in *Intl. Conf. on Pattern Analysis and Intelligent Systems (PAIS)*, 2018, pp. 1–5.

[9] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, March 2018.

[10] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert Systems with Applications*, vol. 165, p. 113679, 2021.

[11] NVIDIA, "Jetson Nano Developer Kit," https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[12] ——, "Tacotron2 and Waveglow 2.0 for PyTorch," https://ngc.nvidia.com/catalog/resources/nvidia:tacotron_2_and_waveglow_for_pytorch/performance.

[13] K. Gimpel, D. Das, and N. A. Smith, "Distributed asynchronous online learning for natural language processing," in *Conf. Computational Natural Language Learning*, 2010.

[14] W. Hemati, T. Uslu, and A. Mehler, "Textimager: a distributed uima-based system for nlp," in *COLING*, 2016.

[15] H. Tang, B. Yin, and R.-H. Wang, "Design of embedded application oriented distributed speech synthesis system with high naturalness," in *Intl. Symp. Chinese Spoken Lang. Process.*, 2002.

[16] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[17] H. Face, "t5-small," https://huggingface.co/t5-small.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[19] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," *arXiv preprint arXiv:1908.08345*, 2019.

[20] C. Tran, "Extractive summarization with bert," https://github.com/chriskhanhtran/bert-extractive-summarization.

[21] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[22] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020.

[23] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.

[24] K. Ito and L. Johnson, "The LJ Speech Dataset," https://keithito.com/LJ-Speech-Dataset/, 2017.

[25] T. Li, "ninmiao/REDIT_test_dataset-0.1," https://doi.org/10.5281/zenodo.5554050, Oct. 2021.