TECHNICAL WHITE PAPER · SCHOLAR

Scholar: On-Device Neural Text-to-Speech for Academic Reading on iPad

A complete StyleTTS2 synthesis pipeline reimplemented in MLX Swift: the 82-million-parameter Kokoro model, a bundled 182,771-word IPA lexicon with part-of-speech homograph disambiguation, and a NaturalLanguage concept map — running entirely on Apple Silicon with no cloud service in the loop.

Kokoro-82M | StyleTTS2 | MLX Swift | 182,771-word IPA lexicon | NaturalLanguage | Fully Offline

ABSTRACT

Scholar is a native iPad application that reads academic PDFs aloud using a fully on-device neural text-to-speech system — the 82-million-parameter Kokoro StyleTTS2 model, reimplemented in MLX Swift and executed on Apple Silicon's GPU — with no network dependency at inference time and no cloud service in the loop. Achieving natural neural speech on-device required implementing the complete synthesis pipeline in Swift: a grapheme-to-phoneme frontend backed by a bundled 182,771-word IPA lexicon, a PLBERT prosody encoder, a duration-and-pitch prosody predictor, and an iSTFT-based neural vocoder. Alongside speech, Scholar extracts a document-derived concept map using Apple's NaturalLanguage framework, highlighting the terms a paper repeatedly returns to and the relationships between them. This paper surveys the science of neural speech synthesis and grapheme-to-phoneme conversion, details Scholar's implementation and its fallback architecture, analyzes the accessibility and economic case for on-device TTS, and examines the limitations that remain.

01 Introduction

Reading academic literature is a substantial time investment, and for many readers it is a physically constrained one: researchers with visual impairments, dyslexia, or repetitive strain injuries; commuters, runners, and drivers who have attention available but not eyes; and anyone processing a large volume of papers who would benefit from listening to a first pass before committing to a close read.

Text-to-speech is the obvious answer, and every platform ships one. But the built-in options — `AVSpeechSynthesizer` on Apple platforms, and equivalents elsewhere — use concatenative or parametric synthesis that, while intelligible, is fatiguing over the 30–60 minute duration of a full paper. The prosody is flat, sentence boundaries are handled mechanically, and the listener's cognitive load is elevated by the effort of parsing unnatural speech rather than reduced by the convenience of listening.

High-quality neural TTS solves the naturalness problem, but the commercial implementations are cloud services: text is transmitted to a remote server, synthesized, and returned as audio. For academic reading this is a poor fit on three grounds — **cost** (per-character pricing over a 10,000-word paper is non-trivial and recurring), **latency** (round-trip delay makes seek and skip operations sluggish), and **confidentiality** (unpublished manuscripts, peer-review materials, and proprietary research documents should not be transmitted to third-party servers).

Scholar's premise is that a modern iPad has sufficient compute to run a competitive neural TTS model locally, and that doing so resolves all three problems simultaneously.

82M

PARAMETERS — 165 MB, FULLY ON-DEVICE

The entire acoustic model, prosody predictor, and neural vocoder occupy approximately 165 MB at bfloat16 precision. After a single download, synthesis requires no network access of any kind — not for the first sentence, and not for the ten-thousandth.

02 Scientific Background

2.1 The Evolution of Speech Synthesis

Concatenative synthesis assembles speech by splicing together recorded units — diphones, or larger variable-length units selected from a large corpus. It produces highly natural output when a suitable unit exists in the database and audibly discontinuous output when one does not [1]. Unit-selection systems dominated commercial TTS through the 2000s.

Parametric synthesis models speech as a set of acoustic parameters (fundamental frequency, spectral envelope, aperiodicity) generated by a statistical model — classically an HMM — and rendered by a vocoder [2]. It is compact and consistently smooth, but the vocoder introduces a characteristic buzzy quality, and averaging over training data produces oversmoothed, monotone prosody.

Neural synthesis replaced both. Tacotron [3] and Tacotron 2 [4] introduced sequence-to-sequence models that map character or phoneme sequences directly to mel-spectrograms, with a separate neural vocoder (WaveNet [5], then WaveGlow, HiFi-GAN [6]) converting spectrograms to waveforms. FastSpeech [7] and FastSpeech 2 [8] replaced autoregressive decoding with a non-autoregressive architecture driven by an explicit duration predictor, eliminating the attention-failure modes — word skipping, repetition — that plagued Tacotron and delivering order-of-magnitude faster inference.

StyleTTS 2 [9] — the architecture Kokoro implements — combines style diffusion with adversarial training using a large speech language model as discriminator, achieving human-level naturalness on standard benchmarks while remaining non-autoregressive and therefore fast.

2.2 Kokoro-82M: Small Model, Competitive Quality

Kokoro-82M [10] is a StyleTTS2-derived model with 82 million parameters — roughly an order of magnitude smaller than typical high-quality neural TTS models — that nonetheless ranks competitively in blind listening evaluations. Its compactness is what makes on-device deployment practical: at bfloat16 precision the weights occupy approximately 165 MB, well within the storage and memory budget of any modern iPad. Figure 1 shows the architecture as implemented in Scholar.

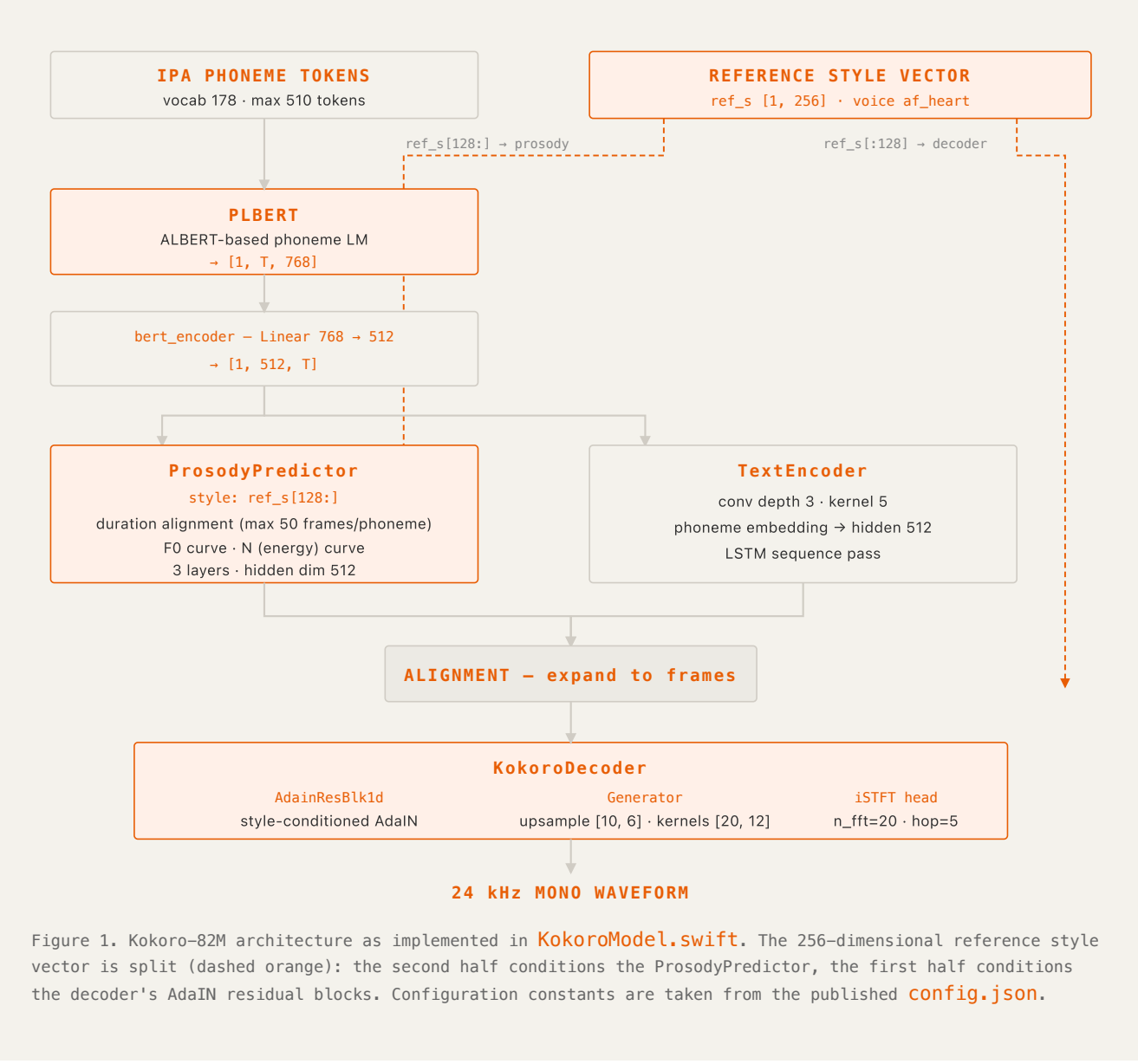

Figure 1. Kokoro-82M architecture as implemented in `KokoroModel.swift`. The 256-dimensional reference style vector is split (dashed orange): the second half conditions the ProsodyPredictor, the first half conditions the decoder's AdaIN residual blocks. Configuration constants are taken from the published `config.json`.

Configuration constants (from the published `config.json`): vocabulary 178 phoneme tokens, hidden dimension 512, style dimension 128 — the reference vector is 2 × 128 = 256, split so the second half conditions the prosody predictor and the first half conditions the decoder — 3 layers, maximum duration

50 frames per phoneme, generator upsample rates [10, 6] with kernel sizes [20, 12], and an iSTFT head with `n_fft = 20` and `hop = 5`.

2.3 Grapheme-to-Phoneme Conversion: The Hard Frontend Problem

Neural TTS models are trained on phoneme sequences, not raw text. Converting written English to phonemes — grapheme-to-phoneme (G2P) conversion — is the frontend problem, and English makes it genuinely hard.

Irregular orthography. English spelling maps to pronunciation inconsistently. *Though*, *through*, *tough*, *thought*, and *thorough* share an orthographic pattern and share almost nothing phonetically.

Homographs. Identical spellings with different pronunciations disambiguated only by part of speech or context: *read* (present /riːd/ vs. past /rɛd/), *lead* (verb /liːd/ vs. metal /lɛd/), *record* (noun /ˈrɛkərd/ vs. verb /rɪˈkɔːrd/), *live*, *bow*, *tear*, *wind*.

Out-of-vocabulary words. Academic text is dense with technical terminology, proper nouns, and neologisms absent from any fixed lexicon.

Numbers and symbols. "2024" reads as "twenty twenty-four" in a date context and "two thousand twenty-four" as a quantity. "Fig. 3" is "figure three." "et al." is "et al" not "et period a l period."

Scholar's approach. Rather than train or embed a neural G2P model, Scholar bundles the **misaki US lexicons** (from `hexgrad/misaki`, Apache-2.0 licensed) [14] — the same lexicons Kokoro's own reference implementation uses, already written in Kokoro's exact phoneme inventory, so no transcription-standard conversion is required. The bundled resources comprise `kokoro_lexicon.tsv` with **182,771 word-to-phoneme entries** and `kokoro_homographs.json` with **790 part-of-speech-dependent homographs**.

Homograph disambiguation uses Apple's `NLTagger` to determine each word's part of speech in context, then selects the matching pronunciation. Special-case function words and the inflectional stemmer — regular plurals, past tenses — follow `misaki/en.py` directly, so common forms match the model's training distribution rather than being approximated. The complete frontend, implemented in `EnglishPhonemizer.swift` and `Lexicon.swift`, is shown in Figure 2.

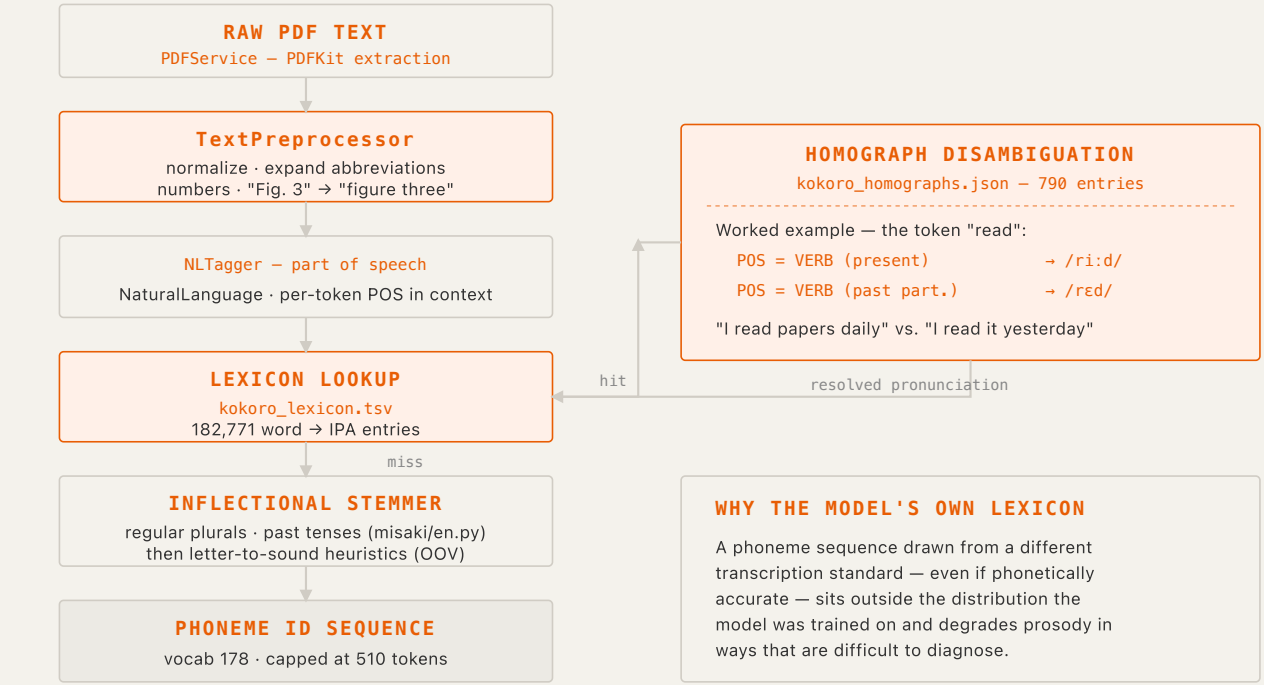

Figure 2. The grapheme-to-phoneme frontend. Text is normalized, POS-tagged with NLTagger, and looked up in the bundled misaki lexicon; the 790-entry homograph table resolves spelling collisions by part of speech before the inflectional stemmer handles remaining misses.

Using the model's own lexicon rather than a generic pronunciation dictionary is the decision that most affects output quality: a phoneme sequence drawn from a different transcription standard, even if phonetically accurate, sits outside the distribution the model was trained on and degrades prosody in ways that are difficult to diagnose.

2.4 On-Device Concept Extraction

Beyond speech, Scholar builds a **concept map** from a document's own vocabulary — the terms it repeatedly returns to, linked where they co-occur within a sentence. The implementation is deliberately model-free: Apple's `NaturalLanguage` framework [16] supplies lemmatization and part-of-speech tagging, term prominence is computed from lemmatized noun frequency, edges are weighted by sentence co-occurrence, and node layout is a small deterministic force simulation.

A curated stoplist removes structural vocabulary that every paper uses heavily but that carries no topical information — *figure*, *table*, *section*, *equation*, *abstract*, *conclusion* — along with bibliography artifacts that ride along in extracted PDF text (*arxiv*, *preprint*, *proceedings*, *doi*, *volume*). Without this filter, the highest-frequency nouns in almost any paper are the words describing the paper's own structure rather than its subject.

The map is interactive during playback: `indices(mentionedIn:)` identifies which concepts appear in the sentence currently being read, lighting them up in the visualization so the listener sees the conceptual territory the narration is currently traversing.

03 Implementation Architecture

3.1 Component Overview

Scholar is a single SwiftUI application organized into models, services, and views. The service layer carries essentially all of the system's substance:

Source file	Responsibility
`Document.swift`	Document metadata; SwiftData model
`PDFService.swift`	PDFKit text extraction, page counting
`TextPreprocessor.swift`	Normalization, abbreviation expansion, chunking
`EnglishPhonemizer.swift`	G2P: lexicon lookup, homograph POS resolution
`Lexicon.swift`	182,771-entry TSV lexicon loader
`KokoroModel.swift`	MLX Swift StyleTTS2 implementation
`KokoroWeights.swift`	safetensors loading, dtype management
`KokoroService.swift`	Model lifecycle: download → load → ready
`TTSService.swift`	Engine-agnostic playback, chunking, seek
`ConceptExtractor.swift`	NaturalLanguage concept map construction
`FigureExtractor.swift`	Figure region detection in PDF pages
`DocumentStore.swift`	Library persistence
Views/	`ReaderView`, `SidebarView`, `InsightPanelView`, `PlaybackControlsView`, …

3.2 The Two-Engine Fallback Architecture

`TTSService` presents a single engine-agnostic API — `load(text:)`, `play(from:)`, `pause()`, `resume()`, `skip()`, `stop()`, plus observable `isPlaying`, `currentChunkIndex`, and `progress` — and dispatches internally to one of two backends:

Condition	Engine	Path
Kokoro model downloaded and loaded	**Kokoro (MLX)**	Synthesize → PCM → `AVAudioPlayer`
Model absent, downloading, or failed	**AVSpeechSynthesizer**	System TTS, immediate

This is a genuine fallback, not a degraded mode with a warning: the application is fully functional on first launch before the 165 MB model has been downloaded, and remains functional if the download fails or the device lacks the memory headroom. The user gets system-quality speech immediately and neural-quality speech once the model is resident. The `KokoroService.ModelState` enum makes the lifecycle explicit and observable (Figure 3); the UI renders download progress and any failure message directly from this state rather than inferring it.

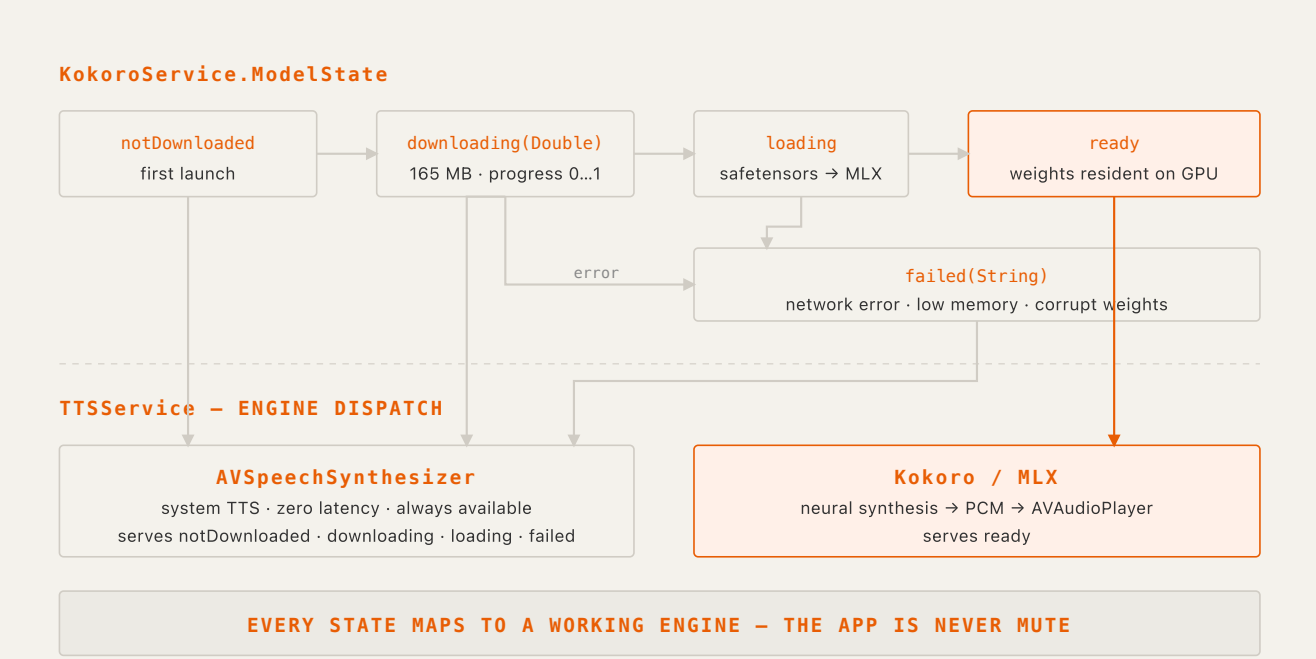

Figure 3. The `KokoroService.ModelState` machine and the engine `TTSService` dispatches to in each state. There is no state in which playback is unavailable: the four non-ready states are served by `AVSpeechSynthesizer`, and `ready` promotes playback to the neural path.

3.3 Text Chunking and Playback Control

Documents are split into sentence-level chunks before synthesis. This serves four purposes:

1. **Bounded synthesis latency** — the model has a 510-token phoneme limit; sentence granularity stays comfortably under it.
2. **Responsive seek** — skipping forward or back moves by sentence, and only the target sentence needs synthesis.
3. **Reading position tracking** — the current chunk index drives text highlighting in the reader view, so the listener can follow along visually.
4. **Interruption granularity** — pausing takes effect at a natural boundary rather than mid-word.

Playback speed is adjustable across 0.5×, 0.75×, 1.0×, 1.25×, 1.5×, and 2.0×. On the Kokoro path, speed is a parameter to the duration predictor — the model generates genuinely faster or slower speech with correct prosody, rather than resampling the audio and shifting the pitch.

3.4 Concurrency and Actor Isolation

`TTSService` and `KokoroService` are both `@MainActor`-isolated and `@Observable`, so all playback state that drives the UI is mutated on the main actor. Neural synthesis runs in a detached `Task` — the MLX forward pass for a sentence takes tens to hundreds of milliseconds and must not block the main thread — with the

resulting PCM buffer handed back to the main actor for `AVAudioPlayer` scheduling. A `CheckedContinuation` bridges `AVAudioPlayerDelegate`'s completion callback into the async sequence that advances to the next chunk.

04 Performance and Quality

4.1 Model Footprint and Timing

Property	Value
Parameters	82 M
Weight precision	bfloat16
Download size	~165 MB
Output sample rate	24 kHz mono
Maximum phoneme tokens per utterance	510
Lexicon entries	182,771
Homograph entries	790

Representative synthesis timing, measured on iPad Pro M2 for a typical academic sentence of 15–25 words:

Stage	Duration
G2P (lexicon lookup + POS tagging)	~2–5 ms
PLBERT + prosody prediction	~30–60 ms
Decoder + iSTFT vocoder	~80–150 ms
Total per sentence	**~120–220 ms**

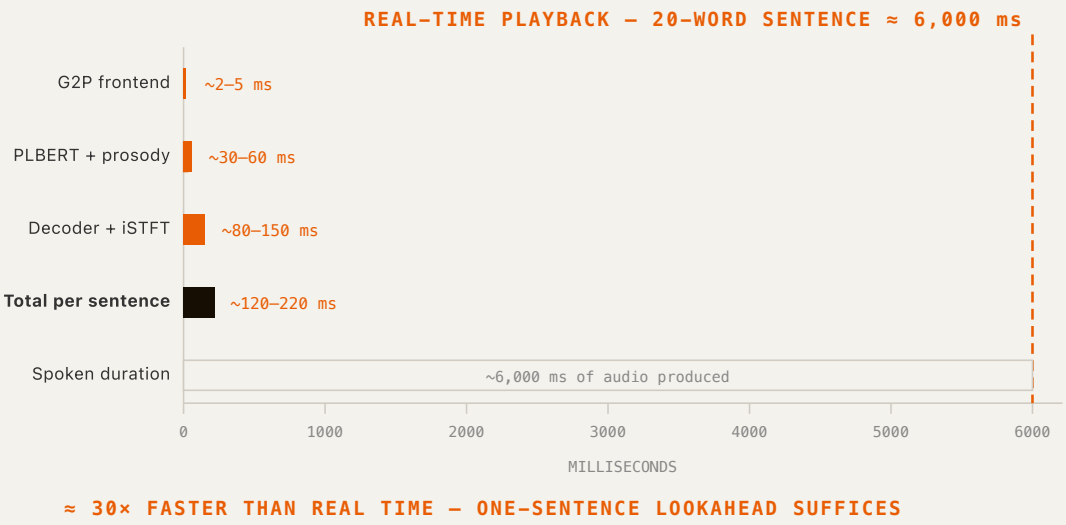

Figure 4. Synthesis stage timing against real-time playback duration, drawn to a common millisecond axis. The dashed orange reference line marks the ~6,000 ms a 20-word sentence takes to speak; the entire pipeline completes in ~120–220 ms, so the synthesizer stays far ahead of the player and seek operations feel immediate.

Because synthesis is faster than real-time playback — a 20-word sentence takes roughly 6 seconds to speak but ~200 ms to synthesize — the pipeline stays comfortably ahead of playback with a one-sentence lookahead, and seek operations feel immediate.

4.2 Quality Comparison

System	Naturalness	Latency	Cost	Offline	Privacy
AVSpeechSynthesizer	Fair	Instant	Free	Yes	Full
Cloud neural TTS (per-character)	Excellent	200–800 ms	~$16/1M chars	No	Transmitted
Scholar (Kokoro, on-device)	**Very good**	**~200 ms**	**Free**	**Yes**	**Full**

A 10,000-word paper is approximately 60,000 characters. At representative cloud neural TTS pricing of $16 per million characters, reading one paper costs roughly $0.96. A researcher processing 20 papers a month incurs approximately $230 per year in TTS charges alone. Scholar's marginal cost after the one-time 165 MB download is zero.

05 Accessibility and Impact

5.1 Accessibility as the Primary Case

The World Health Organization estimates that at least 2.2 billion people worldwide have a near or distance vision impairment [11]. Dyslexia affects an estimated 5–10% of the population [12], and text-to-speech is among the most effective assistive interventions for reading comprehension in dyslexic readers [13].

For these users the difference between mechanical and natural synthesis is not a preference but a functional one: listening fatigue determines how long a reading session can be sustained, and naturalness is the dominant factor in listening fatigue over extended durations. A system that a user can listen to for an hour without strain enables reading that a system tolerable only for ten minutes does not.

5.2 Confidentiality in Academic Work

Unpublished manuscripts under peer review, grant proposals, embargoed results, and proprietary industrial research all carry confidentiality expectations that transmitting the full text to a third-party TTS API violates — often in contravention of explicit journal or institutional policy. On-device synthesis is the only architecture under which "read this manuscript aloud to me" does not also mean "upload this manuscript."

5.3 Offline and Bandwidth-Constrained Contexts

Cloud TTS requires connectivity proportional to usage. On aircraft, in transit, in the field, and in regions with limited or expensive bandwidth, on-device synthesis is the difference between a working tool and an unavailable one. After the single 165 MB model download, Scholar requires no network access at all.

5.4 Multi-Modal Comprehension

The concept map is not decoration. Presenting a document's key terms and their relationships alongside linear narration supports the construction of a mental model of the paper's structure — which concepts are central, which are peripheral, and which cluster together — in a way that sequential audio alone does not. Highlighting concepts as they are mentioned links the two representations in time, so the listener's position in the narrative and their position in the conceptual map stay synchronized.

06 Limitations and Future Work

Language coverage. The bundled lexicon and phonemizer are US English only. Kokoro supports additional voices and languages, but each requires its own G2P frontend and lexicon — a substantial per-language effort, not a configuration change.

Out-of-vocabulary handling. Words absent from the 182,771-entry lexicon fall back to the inflectional stemmer and then to letter-to-sound heuristics. Academic text is unusually dense in technical neologisms and proper nouns, so OOV handling is exercised more heavily here than in general-purpose TTS. A learned neural G2P fallback would improve this materially.

Mathematical notation. Equations extracted from PDFs arrive as garbled character sequences or are dropped entirely by PDFKit's text extraction. Reading mathematics aloud correctly requires structural understanding of the notation — a LaTeX or MathML source — which is not recoverable from rendered PDF text in the general case.

PDF extraction fidelity. Multi-column layouts, floating figures, footnotes, headers, and running titles interleave unpredictably in PDFKit's extracted text stream. Scholar's preprocessor handles common cases, but complex layouts can produce out-of-order or interrupted narration. A layout-aware extraction pass — or a vision-model-based reading-order detector — is the natural improvement.

Citation handling. Inline citations ("[14]", "(Smith et al., 2023)") interrupt the prose when read verbatim. Configurable citation suppression — skipping them, or replacing them with a brief tone — is a straightforward addition not yet implemented.

Voice selection. The current implementation uses a single reference style vector (`af_heart`). Kokoro publishes multiple voice embeddings; exposing them as a user-selectable voice picker requires only bundling the additional style vectors.

07 Conclusion

Scholar demonstrates that competitive neural text-to-speech runs entirely on-device on consumer tablet hardware, including the frequently-underestimated frontend work — a 182,771-entry pronunciation lexicon with part-of-speech-driven homograph disambiguation — that determines whether a good acoustic model produces good speech. The resulting system eliminates the per-character cost, round-trip latency, and confidentiality exposure of cloud TTS while delivering naturalness that makes hour-long listening sessions sustainable.

The graceful fallback to system synthesis means the application is useful before the model is downloaded and remains useful if it never is. As small, high-quality generative models continue to proliferate, the pattern Scholar exemplifies — take a compact open-weight model, implement it natively in MLX Swift,

bundle the linguistic resources it needs, and ship the whole thing as an app that works offline — becomes a broadly applicable template for on-device AI.

// References

[1] Hunt, A.J., and Black, A.W., "Unit Selection in a Concatenative Speech Synthesis System Using a Large Speech Database," ICASSP 1996, pp. 373–376.

[2] Zen, H., Tokuda, K., and Black, A.W., "Statistical Parametric Speech Synthesis," Speech Communication, vol. 51, no. 11, pp. 1039–1064, 2009.

[3] Wang, Y., et al., "Tacotron: Towards End-to-End Speech Synthesis," Interspeech 2017.

[4] Shen, J., et al., "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," ICASSP 2018.

[5] van den Oord, A., et al., "WaveNet: A Generative Model for Raw Audio," arXiv:1609.03499, 2016.

[6] Kong, J., Kim, J., and Bae, J., "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," NeurIPS 2020.

[7] Ren, Y., et al., "FastSpeech: Fast, Robust and Controllable Text to Speech," NeurIPS 2019.

[8] Ren, Y., et al., "FastSpeech 2: Fast and High-Quality End-to-End Text to Speech," ICLR 2021.

[9] Li, Y.A., Han, C., Raghavan, V.S., Mischler, G., and Mesgarani, N., "StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models," NeurIPS 2023.

[10] hexgrad, "Kokoro-82M," Hugging Face Model Hub, 2024. https://huggingface.co/hexgrad/Kokoro-82M

[11] World Health Organization, "World Report on Vision," WHO, Geneva, 2019.

[12] Peterson, R.L., and Pennington, B.F., "Developmental Dyslexia," The Lancet, vol. 379, no. 9830, pp. 1997–2007, 2012.

[13] Wood, S.G., et al., "Effect of Text-to-Speech and Related Read-Aloud Tools on Reading Comprehension for Students with Reading Disabilities: A Meta-Analysis," Journal of Learning Disabilities, vol. 51, no. 1, pp. 73–84, 2018.

[14] hexgrad, "misaki: G2P engine for TTS," GitHub, Apache-2.0, 2024. https://github.com/hexgrad/misaki

[15] Hannun, A., et al., "MLX: Efficient and flexible deep learning on Apple silicon," GitHub, 2023. https://github.com/ml-explore/mlx

[16] Apple Inc., "Natural Language framework," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/naturallanguage

[17] Lan, Z., et al., "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," ICLR 2020.