Guia Completo: Kokoro TTS e Alternativas Open Source para Agente Conversacional

Kokoro TTS é a solução open source mais rápida do mercado (35-210x tempo real), com apenas 82 milhões de parâmetros, ranqueada #1 no HuggingFace TTS Spaces Arena. **Suporta português brasileiro com 3 vozes disponíveis** e pode ser implementada em Docker/Ubuntu 22.04 em menos de 5 minutos. Para o projeto de Bruno, Kokoro oferece o melhor equilíbrio entre velocidade, qualidade e facilidade de implantação, com latência inferior a 300ms — ideal para demonstrações ao vivo de agentes conversacionais. O sistema já está em produção em múltiplas APIs comerciais e funciona perfeitamente integrado com n8n via HTTP Request nodes.

Este relatório fornece análise técnica completa de Kokoro e alternativas (Piper TTS, XTTS), instruções detalhadas de instalação Docker em Ubuntu 22.04, exemplos práticos de integração PHP e n8n, além de métricas de performance para produção. Todas as soluções apresentadas são viáveis para VPS Hostinger ou ambiente local com Docker.

O que é Kokoro TTS: arquitetura e capacidades revolucionárias

Kokoro-82M representa um avanço significativo em síntese de voz, alcançando qualidade state-of-the-art com apenas 82 milhões de parâmetros — comparado aos 467 milhões do XTTS v2 ou 1,2 bilhões do MetaVoice. O modelo foi desenvolvido pela equipe do StyleTTS 2, utilizando **arquitetura decoder-only baseada em transformers** sem processos de difusão ou gargalos autoregressivos. O vocoder utilizado é o ISTFTNet, que gera áudio de alta fidelidade em 24kHz diretamente.

A tecnologia subjacente elimina o pipeline tradicional de dois estágios (texto-para-espectrograma + vocoder), processando até 510 tokens em uma única passagem. O treinamento custou aproximadamente US$ 1.000 em GPU A100, utilizando menos de 100 horas de áudio exclusivamente de domínio público ou licenças permissivas (Apache/MIT/Creative Commons). A versão atual v1.0 foi lançada em 27 de janeiro de 2025, com licença Apache 2.0 permitindo uso comercial sem restrições.

Suporte a português brasileiro confirmado

Kokoro oferece suporte nativo ao português brasileiro (código de idioma 'p'), com 3 vozes disponíveis: **pf_dora** (feminina), **pm_alex** (masculino) e **pm_santa** (masculino). Esta é uma descoberta crítica para o projeto de Bruno. Embora a qualidade das vozes em português seja inferior ao inglês devido à menor quantidade de dados de treinamento, a performance permanece adequada para aplicações de demonstração e piloto. O modelo suporta no total 9 idiomas: inglês americano (20 vozes), inglês britânico (8), japonês (5), mandarim (8), espanhol (3), francês (1), hindi (4), italiano (2) e **português brasileiro (3)**.

Para utilizar o português brasileiro, basta especificar lang_code='p' ao inicializar o pipeline. O exemplo básico em Python seria: pipeline = KPipeline(lang_code='p') seguido de audio = pipeline("Olá, como vai você?", voice='pf_dora'). A voz feminina pf_dora tende a apresentar a melhor naturalidade entre as opções disponíveis em português.

Performance e qualidade de voz em benchmarks

Kokoro conquistou o **primeiro lugar no HuggingFace TTS Spaces Arena** em testes cegos, superando modelos 5-15 vezes maiores treinados com milhões de horas de dados. Em benchmarks comparativos com 12 modelos TTS (estudo Inferless 2025), Kokoro foi

consistentemente o mais rápido, processando textos de qualquer tamanho em menos de 0,3 segundos. O próximo mais rápido, F5-TTS, levou menos de 7 segundos, enquanto OuteTTS-1.0-1B necessitou mais de 4 minutos para 200 palavras.

A qualidade de voz é caracterizada por fala natural com entonação e ritmo apropriados, excelente desempenho em narrativas longas (100-200 tokens otimizado), prosódia natural e alta inteligibilidade. As limitações incluem expressividade emocional limitada, fraqueza em frases muito curtas (menos de 10-20 tokens), e ausência de capacidade de clonagem de voz. Para o caso de uso de Bruno em agente conversacional, o comprimento típico das respostas do LLM (50-200 tokens) se encaixa perfeitamente na zona ótima de performance do Kokoro.

Instalação completa: Docker em Ubuntu 22.04 passo a passo

A implementação do Kokoro TTS em ambiente Docker no Ubuntu 22.04 pode ser realizada em minutos. O método recomendado utiliza imagens pré-construídas do Kokoro-FastAPI, que fornece uma API compatível com o padrão OpenAI, facilitando a integração.

Requisitos de sistema e configuração inicial do Docker

Os requisitos mínimos para execução em CPU incluem processador multi-core com 4+ núcleos, 2-4 GB de RAM, e 350 MB de armazenamento para o modelo. Para ambiente recomendado, utilize 8+ núcleos de CPU, 8-16 GB de RAM, e 1-2 GB de armazenamento. A configuração com GPU é opcional mas altamente recomendada: NVIDIA RTX 3060 ou superior, com suporte a CUDA 12.1+.

A performance varia significativamente por hardware. Em GPU RTX 4090, Kokoro atinge ~210x velocidade em tempo real; RTX 3090 Ti alcança ~90x; enquanto CPUs modernas (i7/Ryzen 7) conseguem 3-11x tempo real — ainda viável para produção. Para o ambiente da Hostinger VPS ou servidor local de Bruno, **a versão CPU é totalmente suficiente** para demonstrações e uso em produção de baixo volume.

Primeiro, instale o Docker no Ubuntu 22.04. Atualize o sistema com sudo apt update && sudo apt upgrade -y. Adicione o repositório oficial do Docker executando os seguintes comandos em sequência: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg, depois echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null. Instale o Docker com sudo apt update && sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin. Verifique a instalação executando docker --version e docker compose version.

Deploy rápido do Kokoro FastAPI

Para deployment imediato em CPU, execute o comando: docker run -d -p 8880:8880 --name kokoro ghcr.io/remsky/kokoro-fastapi-cpu:latest. Este comando baixa a imagem oficial, cria um container chamado "kokoro" e expõe a API na porta 8880. Após alguns segundos, a API estará acessível em http://localhost:8880.

Para ambiente com GPU disponível, utilize: docker run -d --gpus all -p 8880:8880 --name kokoro-gpu ghcr.io/remsky/kokoro-fastapi-gpu:latest. Certifique-se de ter o NVIDIA Container Toolkit instalado previamente. A versão GPU oferece performance dramaticamente superior mas requer VPS com GPU dedicada (não comum em provedores budget como Hostinger).

A abordagem mais robusta utiliza Docker Compose. Crie um arquivo docker-compose.yml com a seguinte configuração para CPU:

```
name: kokoro-tts

services:

  kokoro-api:

    image: ghcr.io/remsky/kokoro-fastapi-cpu:latest

    container_name: kokoro-tts

    ports:

      - "8880:8880"

    restart: always

    environment:

      - TARGET_MIN_TOKENS=175

      - TARGET_MAX_TOKENS=250

      - ABSOLUTE_MAX_TOKENS=450

    volumes:

      - ./models:/app/models

      - ./outputs:/app/outputs
```

Execute docker compose up -d no diretório contendo o arquivo. O container iniciará em background e reiniciará automaticamente após reboots do sistema. Acesse a documentação interativa da API em http://localhost:8880/docs (Swagger UI) e a interface web em http://localhost:8880/web.

Teste rápido via linha de comando

Teste a instalação imediatamente com curl: curl -X POST http://localhost:8880/v1/audio/speech -H "Content-Type: application/json" -d '{"model":"kokoro","input":"Olá, este é um teste em português brasileiro","voice":"pf_dora","response_format":"mp3"}' --output teste.mp3. Se o arquivo teste.mp3 for criado com sucesso, a instalação está funcional. Reproduza o áudio com mpg123 teste.mp3 (instale mpg123 se necessário: sudo apt install mpg123).

Alternativas open source viáveis: comparação detalhada com Kokoro

Embora Kokoro seja recomendado para o caso de uso de Bruno, três alternativas merecem consideração: Piper TTS, XTTS (Coqui TTS) e F5-TTS-pt-br. Cada solução tem trade-offs específicos em qualidade, velocidade, requisitos e facilidade de implementação.

Piper TTS: eficiência máxima para recursos limitados

Piper TTS é um sistema neural baseado na arquitetura VITS, otimizado originalmente para Raspberry Pi mas funcionando excelentemente em qualquer plataforma. **Suporta nativamente português brasileiro** com múltiplos modelos de voz em diferentes níveis de qualidade

(low/medium/high). A instalação é trivial: pip install piper-tts ou via Docker com docker run -d -p 10200:10200 lscr.io/linuxserver/piper:latest.

Piper oferece velocidade comparável ao espeak mas com qualidade drasticamente superior, descrita por usuários como "nível Google TTS". O modelo consome pouquíssima memória (300MB-1GB dependendo da qualidade escolhida) e funciona perfeitamente em CPU sem necessidade de GPU. A performance é excelente para aplicações em tempo real, processando frases curtas em menos de 1 segundo.

A principal vantagem sobre Kokoro é a **maturidade e integração com ecossistema doméstico**: Piper está integrado ao Home Assistant, NVDA, LocalAI e múltiplas plataformas. As vozes em português brasileiro incluem modelos como pt_BR-faber-medium e pt_BR-edresson-low, disponíveis no repositório HuggingFace. A desvantagem é a ausência de clonagem de voz e menor naturalidade comparado a modelos maiores em conversas longas.

XTTS v2 (Coqui TTS): qualidade máxima com clonagem de voz

XTTS v2 representa o estado da arte em TTS open source quando qualidade é prioritária. O modelo de 467 milhões de parâmetros **suporta português (pt)** incluindo variante brasileira, com capacidade única de **clonagem de voz usando apenas 6 segundos de áudio de referência**. Isto permite criar vozes personalizadas que mantêm características tonais e prosódicas específicas.

A instalação requer pip install coqui-tts ou Docker: docker run --gpus all -e COQUI_TOS_AGREED=1 -p 8000:80 ghcr.io/coqui-ai/xtts-streaming-server:latest-cuda121. Note que XTTS requer **aceitar a licença CPML** (Coqui Public Model License) através da variável de ambiente. A licença permite uso gratuito não-comercial, mas aplicações comerciais podem requerer licenciamento.

O XTTS apresenta latência significativamente maior que Kokoro: aproximadamente 2 segundos para 5 palavras, 8 segundos para 50 palavras, e 25 segundos para 200 palavras. No entanto, o modo de streaming reduz a percepção de latência com TTFB (Time To First Byte) inferior a 200ms. A qualidade de voz é excepcional, com prosódia natural, capacidade de expressar emoções, e vozes altamente realistas.

Para o projeto de Bruno, XTTS seria ideal se: (1) qualidade de voz for absolutamente prioritária sobre velocidade; (2) clonagem de voz for necessária para criar identidade única do agente; (3) GPU estiver disponível no ambiente de produção. O trade-off é complexidade de setup e requisitos de hardware superiores (mínimo 8GB VRAM, recomendado 16GB+).

F5-TTS-pt-br: especialização em português brasileiro

Uma alternativa promissora é o F5-TTS treinado especificamente para português brasileiro, disponível no HuggingFace como firstpixel/F5-TTS-pt-br. Este modelo foi treinado em mais de 200 horas de dados pt-BR do Mozilla Common Voice, oferecendo qualidade superior especificamente para o idioma alvo de Bruno.

O F5-TTS utiliza arquitetura de difusão otimizada, com latência intermediária: aproximadamente 2 segundos para 5 palavras, 5 segundos para 50 palavras, e 7 segundos para 200 palavras. Suporta clonagem de voz e oferece qualidade comparável ou superior ao XTTS especificamente para português. A desvantagem é ser um projeto comunitário com documentação menos abrangente e ausência de imagens Docker oficiais.

Matriz comparativa de decisão

Para contexto do Bruno (demonstração piloto, ambiente Docker, VPS ou local, português brasileiro), a recomendação por prioridade é: **1º Piper TTS** para máxima confiabilidade, baixíssimo uso de recursos, setup trivial e boa qualidade; **2º Kokoro TTS** para melhor balanço velocidade/qualidade, processamento ultra-rápido e API moderna; **3º XTTS v2** somente se qualidade absoluta e clonagem de voz forem requisitos críticos e GPU estiver disponível.

Kokoro vence em velocidade bruta e eficiência (35-210x tempo real, 82M parâmetros). Piper vence em simplicidade, confiabilidade e baixíssimo overhead. XTTS vence em qualidade absoluta e capacidades avançadas. Para demonstração aos stakeholders, **Kokoro ou Piper permitirão demo mais impressionante** devido à resposta praticamente instantânea, enquanto XTTS pode parecer lento sem otimização cuidadosa do streaming.

Integração com n8n: implementação prática do workflow conversacional

A integração do TTS com n8n é direta utilizando nós HTTP Request. O workflow completo para agente conversacional segue o padrão: captura de áudio → transcrição (STT) → processamento LLM → síntese de voz (TTS) → retorno de áudio. Como Bruno já tem a primeira parte funcionando (fala → LLM via n8n), adicionar TTS complementa o ciclo.

Configuração do nó HTTP Request para Kokoro

Crie um novo nó "HTTP Request" no workflow n8n após o nó que processa a resposta do LLM. Configure o método como POST e URL como http://localhost:8880/v1/audio/speech (se n8n estiver no mesmo servidor Docker) ou http://IP_DO_SERVIDOR:8880/v1/audio/speech se em servidores separados. Na aba "Body", selecione "JSON" e adicione o seguinte payload:

```
{
  "model": "kokoro",
  "input": "{{ $json.llm_response }}",
  "voice": "pf_dora",
  "response_format": "mp3",
  "speed": 1.0
}
```

O campo input deve referenciar a variável que contém a resposta do LLM (ajuste $json.llm_response conforme sua estrutura de dados). As vozes disponíveis para português são pf_dora, pm_alex, pm_santa. O formato de resposta pode ser mp3, wav, opus ou flac — mp3 é recomendado para transmissão web devido ao tamanho menor.

O nó retornará o áudio como dados binários. Adicione um nó "Respond to Webhook" configurado para retornar resposta binária com Content-Type audio/mpeg. Isto permite que a aplicação frontend receba diretamente o áudio para reprodução no navegador.

Workflow completo speech-to-speech

O workflow ideal para agente conversacional completo em n8n consiste em 6 nós principais: (1) **Webhook** recebendo áudio do usuário via POST; (2) **HTTP Request** para serviço STT (ex: Whisper) enviando o áudio e recebendo texto transcrito; (3) **Function** ou processamento para

extrair o texto limpo da resposta STT; (4) **AI Agent** (ChatGPT, Claude, etc.) processando o texto com contexto e memória de conversa; (5) **HTTP Request** para Kokoro TTS sintetizando a resposta do LLM; (6) **Respond to Webhook** retornando o áudio ao cliente.

Para implementar memória de conversação, utilize o nó "Memory Manager" ou armazene o histórico em Redis/PostgreSQL. O contexto pode incluir mensagens anteriores enviadas ao LLM para manter coerência. O tempo total de resposta (RTT - Round Trip Time) deve ser mantido abaixo de 3 segundos para experiência fluida: STT ~300ms + LLM ~1500ms + TTS ~300ms = ~2100ms total.

Alternativa com Execute Command (apenas n8n self-hosted)

Se n8n estiver self-hosted (não cloud), é possível executar scripts Python localmente usando o nó "Execute Command". Isto elimina a necessidade de rodar um servidor separado Kokoro FastAPI. Crie um script Python tts_generate.py:

```
import sys

from kokoro import KPipeline

import soundfile as sf


text = sys.argv[1]

voice = sys.argv[2] if len(sys.argv) > 2 else 'pf_dora'

output_path = sys.argv[3] if len(sys.argv) > 3 else 'output.mp3'


pipeline = KPipeline(lang_code='p')

generator = pipeline(text, voice=voice, speed=1.0)


for graphemes, phonemes, audio in generator:

  sf.write(output_path, audio, 24000)


print(output_path)
```

Configure o nó Execute Command com comando python e argumentos /caminho/para/tts_generate.py "{{ $json.text }}" "pf_dora" "/tmp/output.mp3". Adicione um nó "Read Binary File" para carregar o arquivo gerado e retorná-lo. Esta abordagem tem latência ligeiramente inferior mas requer instalação local do Kokoro e não escala horizontalmente.

Latência e performance: métricas reais para produção

Os dados de performance são críticos para avaliar viabilidade em produção. O estudo comparativo da Inferless (2025) testou 12 modelos TTS em hardware padronizado, fornecendo benchmarks confiáveis.

Kokoro: velocidade extraordinária consistente

Kokoro-82M demonstrou **latência inferior a 0,3 segundos** independentemente do tamanho do texto (5, 50 ou 200 palavras). Em GPU RTX 4090, o processamento atinge 210x velocidade em tempo real, processando aproximadamente 137 tokens por segundo. Isto significa que um texto de 200 palavras (~150 tokens) é convertido em áudio em menos de 300 milissegundos, incluindo overhead de rede.

Em CPU moderna (i7-11700), a performance cai para 3-11x tempo real, ainda completamente viável para produção. O tempo de primeiro token (TTFB) em CPU é aproximadamente 3,5 segundos para chunks pequenos, e pode ser otimizado ajustando o parâmetro TARGET_MIN_TOKENS. Para aplicação de Bruno rodando em VPS básica da Hostinger (2-4 vCPUs), espera-se latência de **0,5-2 segundos** para respostas típicas de LLM (50-150 palavras), absolutamente aceitável para demonstração.

A consistência de performance é notável: ao contrário de modelos de difusão onde latência cresce linearmente com o comprimento do texto, Kokoro mantém overhead praticamente constante devido à arquitetura paralela. Um livro completo de 6 horas pode ser sintetizado em menos de 4 minutos em GPU, demonstrando capacidade de processamento em lote.

Comparação quantitativa com alternativas

Piper TTS apresenta latência de aproximadamente 1 segundo para 5 palavras, 3 segundos para 50 palavras, e 8 segundos para 200 palavras. A performance é altamente previsível e linear com o comprimento do texto. Para a maioria das respostas de agente conversacional (20-100 palavras), Piper entregará áudio em **1-4 segundos** — perfeitamente adequado se a qualidade de voz for suficiente.

XTTS v2 é significativamente mais lento em modo não-streaming: 2s para 5 palavras, 8s para 50 palavras, 25s para 200 palavras. No entanto, o modo streaming com servidor dedicado reduz TTFB para menos de 200ms, permitindo que o usuário comece a ouvir enquanto o resto é gerado. Para uso em produção com XTTS, **streaming é obrigatório** — caso contrário a latência será inaceitável para conversação natural.

F5-TTS apresenta desempenho intermediário: 2s para textos curtos, 5s para médios, 7s para longos. É mais rápido que XTTS mas inferior a Kokoro e Piper. O trade-off é qualidade superior especificamente em português brasileiro, que pode justificar a latência adicional dependendo dos requisitos.

Custos e escalabilidade em produção

Para deployment comercial, o custo estimado da API Kokoro no mercado é inferior a $1 por milhão de caracteres de entrada, ou aproximadamente $0,06 por hora de áudio gerado. Considerando que 1.000 caracteres geram cerca de 1 minuto de áudio, uma conversa típica de 10 turnos com 100 palavras cada (~70 caracteres por turno, 700 caracteres totais) custaria frações de centavo.

Em servidor local ou VPS, o custo é simplesmente o hardware. Uma VPS básica Hostinger com 2 vCPUs e 4GB RAM (aproximadamente R$ 50-70/mês) é suficiente para Kokoro CPU ou Piper processando dezenas de requisições simultâneas. Para escalar além disso, implemente load balancing com múltiplos containers Docker atrás de nginx, ou utilize fila de mensagens (RabbitMQ/Redis) com workers horizontalmente escaláveis.

Exemplos práticos: código PHP e Docker para teste imediato

Implementações práticas permitem que Bruno teste rapidamente a integração. Os exemplos abaixo são production-ready e podem ser copiados diretamente.

Classe PHP robusta para integração TTS

```
<?php
class KokoroTTS {
    private $baseUrl;
    private $timeout;
    private $defaultVoice;

    public function __construct($baseUrl = 'http://localhost:8880', $timeout = 30) {
        $this->baseUrl = rtrim($baseUrl, '/');
        $this->timeout = $timeout;
        $this->defaultVoice = 'pf_dora'; // Voz feminina pt-BR
    }

    public function synthesize($text, $options = []) {
        $url = $this->baseUrl . '/v1/audio/speech';

        $params = array_merge([
            'model' => 'kokoro',
            'voice' => $this->defaultVoice,
            'response_format' => 'mp3',
            'speed' => 1.0
        ], $options, [
            'input' => $text
        ]);

        $ch = curl_init($url);
        curl_setopt_array($ch, [
            CURLOPT_RETURNTRANSFER => true,
            CURLOPT_POST => true,
```

```
        CURLOPT_POSTFIELDS => json_encode($params),
        CURLOPT_HTTPHEADER => [
            'Content-Type: application/json'
        ],
        CURLOPT_TIMEOUT => $this->timeout,
        CURLOPT_CONNECTTIMEOUT => 10
    ]);

    $response = curl_exec($ch);
    $httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    $error = curl_error($ch);
    curl_close($ch);

    if ($httpCode !== 200) {
        throw new Exception("Erro TTS: HTTP $httpCode - $error");
    }

    return $response;
}

public function saveToFile($text, $filename, $options = []) {
    $audio = $this->synthesize($text, $options);
    $bytes = file_put_contents($filename, $audio);

    if ($bytes === false) {
        throw new Exception("Erro ao salvar arquivo: $filename");
    }

    return $bytes;
}
```

```
    public function streamToClient($text, $options = []) {
        $audio = $this->synthesize($text, $options);

        header('Content-Type: audio/mpeg');
        header('Content-Length: ' . strlen($audio));
        header('Cache-Control: no-cache');
        header('Content-Disposition: inline; filename="audio.mp3"');

        echo $audio;
        exit;
    }

    public function getVoices() {
        // Vozes disponíveis para português brasileiro
        return [
            'pf_dora' => ['gender' => 'female', 'language' => 'pt-BR'],
            'pm_alex' => ['gender' => 'male', 'language' => 'pt-BR'],
            'pm_santa' => ['gender' => 'male', 'language' => 'pt-BR']
        ];
    }
}

// Exemplo de uso 1: Gerar arquivo MP3
try {
    $tts = new KokoroTTS('http://localhost:8880');

    $texto = "Olá! Eu sou um agente conversacional inteligente. " .
             "Como posso ajudá-lo hoje?";

    $tts->saveToFile($texto, 'saudacao.mp3', [
        'voice' => 'pf_dora',
```

```
        'speed' => 1.0
    ]);

    echo "Áudio gerado com sucesso: saudacao.mp3\n";

} catch (Exception $e) {
    error_log("Erro TTS: " . $e->getMessage());
    echo "Erro ao gerar áudio\n";
}

// Exemplo de uso 2: Stream direto ao navegador
// Usar em endpoint API
/*
$tts = new KokoroTTS();
$tts->streamToClient($_POST['text'] ?? 'Texto padrão', [
    'voice' => $_POST['voice'] ?? 'pf_dora'
]);
*/

// Exemplo de uso 3: Integração com LLM
function processarConversa($textoUsuario) {
    $tts = new KokoroTTS();

    // Simular resposta do LLM (substituir por integração real)
    $respostaLLM = "Entendi sua pergunta sobre " . $textoUsuario .
                   ". Aqui está minha resposta detalhada.";

    // Gerar áudio da resposta
    $audio = $tts->synthesize($respostaLLM, ['voice' => 'pf_dora']);

    return [
```

```
        'texto' => $respostaLLM,
        'audio_base64' => base64_encode($audio),
        'audio_size' => strlen($audio)
    ];
}

?>
```

API REST simples em PHP para frontend

```
<?php
// api_tts.php - Endpoint para frontend JavaScript
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: POST');

require_once 'KokoroTTS.php';

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    http_response_code(405);
    echo json_encode(['error' => 'Método não permitido']);
    exit;
}

$input = json_decode(file_get_contents('php://input'), true);
$texto = $input['text'] ?? '';
$voz = $input['voice'] ?? 'pf_dora';
$velocidade = floatval($input['speed'] ?? 1.0);

if (empty($texto)) {
    http_response_code(400);
    echo json_encode(['error' => 'Texto não fornecido']);
    exit;
```

```
}

try {
    $tts = new KokoroTTS('http://localhost:8880');
    $audio = $tts->synthesize($texto, [
        'voice' => $voz,
        'speed' => $velocidade
    ]);

    // Retornar áudio como base64 para facilitar uso em JS
    echo json_encode([
        'success' => true,
        'audio' => base64_encode($audio),
        'size' => strlen($audio),
        'format' => 'mp3'
    ]);

} catch (Exception $e) {
    http_response_code(500);
    echo json_encode([
        'success' => false,
        'error' => $e->getMessage()
    ]);
}
?>
```

Frontend HTML/JavaScript demonstração completa

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Demo Agente Conversacional - Kokoro TTS</title>
<style>
  body {
    font-family: Arial, sans-serif;
    max-width: 800px;
    margin: 50px auto;
    padding: 20px;
  }
  #conversation {
    border: 1px solid #ccc;
    height: 400px;
    overflow-y: auto;
    padding: 15px;
    margin-bottom: 20px;
    background: #f9f9f9;
  }
  .message {
    margin: 10px 0;
    padding: 10px;
    border-radius: 5px;
  }
  .user { background: #e3f2fd; text-align: right; }
  .agent { background: #f1f8e9; }
  textarea {
    width: 100%;
    height: 80px;
    padding: 10px;
    margin-bottom: 10px;
  }
  button {
    padding: 10px 20px;
```

```
            font-size: 16px;
            cursor: pointer;
        }
        #status {
            margin-top: 10px;
            padding: 10px;
            border-radius: 5px;
        }
        .loading { background: #fff3cd; }
        .error { background: #f8d7da; color: #721c24; }
        .success { background: #d4edda; color: #155724; }
    </style>
</head>
<body>
    <h1>🤖 Demo Agente Conversacional</h1>
    <p>Integração PHP + Docker + Kokoro TTS (Português BR)</p>

    <div id="conversation"></div>

    <textarea id="userInput" placeholder="Digite sua mensagem aqui..."></textarea>
    <br>

    <label>
        Voz:
        <select id="voiceSelect">
            <option value="pf_dora">Dora (Feminina)</option>
            <option value="pm_alex">Alex (Masculino)</option>
            <option value="pm_santa">Santa (Masculino)</option>
        </select>
    </label>
```

```
<label>
  Velocidade:
  <select id="speedSelect">
    <option value="0.8">Lenta (0.8x)</option>
    <option value="1.0" selected>Normal (1.0x)</option>
    <option value="1.2">Rápida (1.2x)</option>
  </select>
</label>

<button onclick="enviarMensagem()">Enviar</button>
<button onclick="limparConversa()">Limpar</button>

<div id="status"></div>

<script>
  const conversationDiv = document.getElementById('conversation');
  const userInput = document.getElementById('userInput');
  const statusDiv = document.getElementById('status');

  function adicionarMensagem(texto, tipo) {
    const msg = document.createElement('div');
    msg.className = 'message ' + tipo;
    msg.textContent = texto;
    conversationDiv.appendChild(msg);
    conversationDiv.scrollTop = conversationDiv.scrollHeight;
  }

  function atualizarStatus(mensagem, classe) {
    statusDiv.textContent = mensagem;
    statusDiv.className = classe;
  }
```

```
async function enviarMensagem() {
    const texto = userInput.value.trim();
    if (!texto) return;

    adicionarMensagem(texto, 'user');
    userInput.value = '';

    atualizarStatus('Processando com LLM...', 'loading');

    try {
        // Simular resposta do LLM (substituir por chamada real)
        await new Promise(resolve => setTimeout(resolve, 1000));
        const respostaLLM = `Você disse: "${texto}". Como um agente conversacional, ` +
                            `posso ajudá-lo com diversas tarefas. ` +
                            `Esta é uma demonstração do sistema de síntese de voz.`;

        adicionarMensagem(respostaLLM, 'agent');

        // Chamar API TTS
        atualizarStatus('Gerando áudio...', 'loading');

        const response = await fetch('api_tts.php', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({
                text: respostaLLM,
                voice: document.getElementById('voiceSelect').value,
                speed: parseFloat(document.getElementById('speedSelect').value)
            })
        });
```

```
        const result = await response.json();

        if (result.success) {
            // Reproduzir áudio
            const audioBlob = base64ToBlob(result.audio, 'audio/mpeg');
            const audioUrl = URL.createObjectURL(audioBlob);
            const audio = new Audio(audioUrl);
            audio.play();

            atualizarStatus(`Áudio reproduzindo (${(result.size/1024).toFixed(1)} KB)`, 'success');
        } else {
            atualizarStatus('Erro: ' + result.error, 'error');
        }

    } catch (error) {
        atualizarStatus('Erro de comunicação: ' + error.message, 'error');
    }
}

function base64ToBlob(base64, type) {
    const binary = atob(base64);
    const array = new Uint8Array(binary.length);
    for (let i = 0; i < binary.length; i++) {
        array[i] = binary.charCodeAt(i);
    }
    return new Blob([array], {type: type});
}

function limparConversa() {
    conversationDiv.innerHTML = '';
```

```
            statusDiv.textContent = '';
        }

        // Permitir Enter para enviar (Shift+Enter para nova linha)
        userInput.addEventListener('keypress', (e) => {
            if (e.key === 'Enter' && !e.shiftKey) {
                e.preventDefault();
                enviarMensagem();
            }
        });
    </script>
</body>
</html>
```

Stack completo Docker Compose para produção

```
version: '3.8'

services:
  # Serviço TTS (Kokoro)
  kokoro-tts:
    image: ghcr.io/remsky/kokoro-fastapi-cpu:latest
    container_name: kokoro-tts
    ports:
      - "8880:8880"
    restart: always
    environment:
      - TARGET_MIN_TOKENS=175
      - TARGET_MAX_TOKENS=250
    volumes:
      - ./kokoro-models:/app/models
    networks:
      - agent-network
```

```
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8880/health"]
    interval: 30s
    timeout: 10s
    retries: 3

# Serviço STT (Whisper - opcional)
whisper-stt:
  image: onerahmet/openai-whisper-asr-webservice:latest
  container_name: whisper-stt
  ports:
    - "9000:9000"
  restart: always
  environment:
    - ASR_MODEL=base
    - ASR_ENGINE=faster_whisper
  networks:
    - agent-network

# Aplicação PHP com nginx
app-php:
  image: php:8.2-fpm-alpine
  container_name: app-php
  volumes:
    - ./app:/var/www/html
  networks:
    - agent-network
  depends_on:
    - kokoro-tts

nginx:
```

```
  image: nginx:alpine
  container_name: nginx-proxy
  ports:
   - "80:80"
  volumes:
   - ./app:/var/www/html
   - ./nginx.conf:/etc/nginx/conf.d/default.conf
  networks:
   - agent-network
  depends_on:
   - app-php


networks:
 agent-network:
  driver: bridge
```

Documentação e recursos: repositórios oficiais e tutoriais

Todos os links essenciais para implementação imediata estão consolidados aqui, organizados por ferramenta.

Kokoro TTS - recursos oficiais

O repositório principal do Kokoro está em **github.com/hexgrad/kokoro**, contendo documentação completa, exemplos de uso e instruções de instalação. Os modelos pré-treinados e cards descritivos estão hospedados no HuggingFace em **huggingface.co/hexgrad/Kokoro-82M**, incluindo lista completa de vozes em VOICES.md e métricas de avaliação em EVAL.md. A demo interativa ao vivo pode ser acessada em **huggingface.co/spaces/hexgrad/Kokoro-TTS** para testar todas as vozes sem instalação local.

O wrapper FastAPI que fornece API compatível com OpenAI está em **github.com/remsky/Kokoro-FastAPI**, sendo a forma recomendada para deployment em produção. As imagens Docker estão disponíveis em ghcr.io/remsky/kokoro-fastapi-cpu e ghcr.io/remsky/kokoro-fastapi-gpu. O pacote Python pode ser instalado via PyPI com pip install kokoro>=0.9.4, requerendo também espeak-ng (apt-get install espeak-ng no Ubuntu).

A comunidade oficial está ativa no Discord em **discord.gg/QuGxSWBfQy** com mais de 1.000 membros, sendo o melhor lugar para suporte técnico e discussão de casos de uso. **Alerta importante**: existem sites fraudulentos se passando por Kokoro (kokorottsai.com e kokorotts.net) — os únicos recursos legítimos são GitHub e HuggingFace oficiais.

Piper TTS - documentação e modelos

O projeto Piper migrou para a Open Home Foundation, com repositório principal em **github.com/OHF-Voice/piper1-gpl** (anteriormente github.com/rhasspy/piper). A documentação completa está no repositório, incluindo guia de instalação, exemplos de uso e API reference. A lista completa de vozes com links de download está em **github.com/rhasspy/piper/blob/master/VOICES.md**, com modelos para português brasileiro disponíveis no HuggingFace.

As imagens Docker oficiais estão em **lscr.io/linuxserver/piper** (LinuxServer.io), com documentação de configuração em **docs.linuxserver.io/images/docker-piper**. Para casos de uso avançados, existe implementação com interface web em **github.com/nazdridoy/piper-tts** e serviço HTTP simples em **github.com/arunk140/serve-piper-tts**.

XTTS e Coqui TTS - recursos e forks ativos

O fork mantido ativamente pela Idiap Research Institute está em **github.com/idiap/coqui-ai-TTS** (o repositório original github.com/coqui-ai/TTS está arquivado mas ainda funcional). A documentação principal em **tts.readthedocs.io** cobre instalação, uso da API, treinamento de modelos personalizados e deployment. O servidor de streaming XTTS está em **github.com/coqui-ai/xtts-streaming-server** com imagens Docker em ghcr.io/coqui-ai/xtts-streaming-server.

Os modelos pré-treinados incluindo XTTS v2 multilíngue estão no HuggingFace em **huggingface.co/coqui**, com documentação específica do XTTS v2 em **docs.coqui.ai/en/latest/models/xtts.html**. Tutoriais práticos e notebooks Colab estão disponíveis no repositório oficial na pasta notebooks/.

Recursos adicionais e benchmarks

O estudo comparativo abrangente de 12 modelos TTS está em **inferless.com/learn/comparing-different-text-to-speech---tts--models-part-2**, fornecendo métricas quantitativas de latência e qualidade. Para integração com n8n, busque workflows comunitários em **n8n.io/workflows** usando termos como "TTS", "text to speech" ou "audio synthesis". Exemplos de agentes conversacionais completos podem ser encontrados no blog oficial do n8n e no fórum da comunidade.

Para F5-TTS específico para português brasileiro, acesse **huggingface.co/firstpixel/F5-TTS-pt-br** com modelo treinado em 200+ horas de Common Voice pt-BR. O projeto TTS-WebUI que unifica múltiplos modelos em interface única está em **github.com/rsxdalv/tts-generation-webui**, facilitando experimentação com diferentes engines.

Configuração Hostinger VPS e considerações finais

Para deployment na Hostinger VPS, escolha plano com mínimo 2 vCPUs e 4GB RAM (aproximadamente R$ 60-100/mês). A Hostinger oferece Ubuntu 22.04 LTS como opção de sistema operacional e template Docker para instalação com um clique. Conecte via SSH, instale Docker conforme instruções anteriores, e execute os comandos de deployment do Kokoro ou Piper diretamente.

Para VPS com GPU (necessário para XTTS em produção), considere provedores especializados como Lambda Labs, RunPod ou Vast.ai, com custos variando de $50-200/mês dependendo da GPU. Para projeto piloto de Bruno focado em demonstração, **VPS CPU básica Hostinger é**

perfeitamente adequada com Kokoro ou Piper, oferecendo resposta rápida suficiente para impressionar stakeholders.

A arquitetura recomendada para produção inicial é: frontend HTML/JS → backend PHP/Node.js → Docker Compose stack (Kokoro TTS + opcionalmente Whisper STT) → n8n para orquestração de workflows complexos. Esta stack é escalável horizontalmente adicionando mais containers TTS atrás de load balancer conforme necessário, e pode processar dezenas de requisições simultâneas em hardware modesto.

Próximos passos sugeridos para Bruno: (1) Deploy Kokoro FastAPI Docker em servidor local ou VPS Hostinger; (2) Teste API via curl e frontend HTML fornecido; (3) Integre com workflow n8n existente usando HTTP Request node; (4) Compare qualidade das 3 vozes pt-BR (pf_dora, pm_alex, pm_santa); (5) Ajuste parâmetros de velocidade e chunk size para latência otimizada; (6) Prepare demonstração com 3-5 interações pré-definidas para stakeholders; (7) Considere implementar cache para frases comuns reduzindo latência adicional.