ChatGPT

# Modelli TTS Locali per Chiamate in Italiano – Ricerca Completa

## Executive Summary

**Totale modelli trovati:** Oltre **50** modelli open-source e self-hosted di sintesi vocale (TTS) sono stati identificati. Questi variano da semplici motori classici (es. eSpeak) a modelli neurali avanzati rilasciati nel 2024-2025.

**Supporto per l'Italiano:** Circa **20+** modelli supportano esplicitamente la lingua italiana (nativamente o tramite modelli multilingua). Molti altri sono *fine-tunabili* o utilizzabili in italiano con prestazioni variabili. Modelli dedicati esclusivamente all'italiano sono più rari ma includono progetti recenti come **Azzurra-voice** di Cartesia (2 miliardi di parametri, addestrato su migliaia di ore in italiano) e voci open-source basate su dataset italiani (Common Voice IT, M-AILABS IT, ecc.).

**Modelli testati/testabili localmente:** Praticamente tutti i modelli elencati possono essere eseguiti on-premise. Molti dispongono di repository GitHub o pacchetti Hugging Face con istruzioni d'uso. Per i modelli più pesanti (oltre 1-2 miliardi di parametri) l'hardware di riferimento (GPU RTX 3050 Ti 4GB) risulta un limite: alcuni richiedono GPU con più VRAM o ottimizzazioni (quantizzazione, esecuzione su CPU con RAM elevata, ecc.). Nel report si evidenziano requisiti e test di performance noti.

**Top 3 modelli raccomandati (overview):**

- **Coqui XTTS (v2)** – *Multilingua con Voice Cloning Zero-Shot.* Supporta 17 lingue incluso italiano, clonazione vocale con 3-6 secondi di audio [1] [2]. Qualità vocale naturale (MOS stimato ~4/5) e supporto cross-lingua [3]. **Pro:** Alta qualità e flessibilità (zero-shot cloning, emozioni), facile da usare via `TTS` (pip). **Contro:** Pesantezza moderata (circa 1B parametri), licenza *Coqui Public* (richiede verifica uso commerciale). *Consigliato* per applicazioni esigenti (assistenti avanzati) dove la qualità conta più della latenza.

- **Piper (RHVoice/Piper)** – *TTS neurale leggero e veloce.* Ottimizzato per dispositivi embedded (Raspberry Pi) [4], supporta ~20 lingue (voci multiple) tra cui italiano. **Pro:** Installazione semplicissima (`pip` o eseguibile standalone), inferenza rapidissima anche su CPU (latenza ~0.5s per frase breve), licenza MIT/GPL. **Contro:** Qualità variabile a seconda della voce (in italiano suono un po' artificiale, MOS ~3/5). Ottimo per IVR semplici e *use case* con molte chiamate simultanee (fino a decine su CPU multi-core).

- **Resemble Chatterbox** – *TTS 0.5B multilingue, ottimo bilanciamento velocità/qualità.* Supporta 23 lingue tra cui l'italiano [5], con clonazione vocale e controlli espressivi avanzati [6] [7]. **Pro:** Voce molto naturale (Elo elevato su TTS Arena), modulazione emozioni e timbro via prompt testuale, licenza MIT aperta. **Contro:** Richiede GPU per la massima velocità (0.5B parametri), documentazione tecnica essenziale. Ideale come soluzione *general-purpose* per voicebot multilingua con elevata qualità percepita.

**Decision Matrix (Sintesi):**

- *Qualità Italiano:* Migliori modelli (MOS ≥ 4/5) – Azzurra-voice, Coqui XTTS, Resemble Chatterbox, Zonos (se fine-tunato), Parler Multilingual.
- *Latenza/Performance:* Modelli leggeri con RTF > 1× – Piper, Kokoro-82M, Silero V3, eSpeak NG (bassa qualità), Festival/Flite (obsoleti ma velocissimi).
- *Voice Cloning:* Zero-shot top – Coqui XTTS, OpenVoice v2, VALL-E X (ricerca), GPT-SoVITS (zero-shot multi-lingue) [8], Bark (stile), Tortoise (few-shot alta qualità). Fine-tuning efficiente – Parler (nuova voce con pochi minuti di dati), So-VITS, RVC (training <1 ora su voce target).
- *Facilità d'uso:* Più semplici – Piper (una riga di comando, binari precompilati), Coqui TTS (`pip install TTS`), Mimic3 (pacchetto completo), Parler/Transformers (integrazione HuggingFace), Larynx (docker/CLI). Più complesse – ChatTTS (AGPL, va buildato), alcuni progetti di ricerca (richiedono configurazioni manuali, es. Matcha-TTS).
- *Scalabilità 20-50 chiamate:* Approcci – utilizzare modelli leggeri (Piper, Kokoro) su CPU multi-core; oppure pipeline streaming (es. XTTS con streaming 200ms TTFB [9]) con una GPU per più chiamate interlacciate. Analisi TCO indica che l'hosting locale conviene oltre ~5.000-10.000 min/mese rispetto a cloud TTS, a patto di investire in hardware e setup iniziale (ROI ~3-6 mesi, vedi sezione Costi).

**In sintesi**, l'ecosistema TTS open-source nel 2025 offre soluzioni *production-ready* per l'italiano. La scelta ottimale dipende dal **trade-off** fra qualità e performance: per **qualità massima** in italiano si considerino modelli grandi (Azzurra, XTTS, Chatterbox) con una GPU dedicata; per **scalabilità e reattività** meglio modelli leggeri (Piper, Kokoro) o architetture ottimizzate (FastSpeech, GlowTTS) con audio leggermente più sintetico ma latenza minima. Nel seguito del report presentiamo un inventario esaustivo dei modelli e un'analisi dettagliata su qualità linguistica, prestazioni su RTX 3050 Ti, facilità di deployment e considerazioni per le chiamate telefoniche (streaming 8kHz, integrazione VoIP, ecc.).

---

# Sezione 1: Metodologia di Ricerca

## 1.1 Fonti Consultate

Abbiamo condotto una ricerca **esaustiva** su più fronti:

- **Repository di Codice (GitHub, GitLab, Bitbucket):** Utilizzo di keyword *"text to speech"*, *"TTS italian"*, *"voice cloning"*, filtrando per progetti con licenze open (MIT, Apache, GPL) e ultimo update recente. Abbiamo esplorato repository noti (es: `coqui/TTS`, `silero-models`, `mozilla/TTS`, `MycroftAI/mimic3`, `snakers4/silero-models`, ecc.) e progetti emergenti segnalati su Reddit (es: `Zyphra/Zonos`, `ressemble-ai/chatterbox`, `canopyai/Orpheus-TTS`, `huggingface/parler-tts`, `Hexgrad/kokoro`, `2noise/ChatTTS`, `fishaudio/fish-speech`, `myshell-ai/OpenVoice`, etc.). Anche fork e varianti (es. *AllTalk TTS*, *VoiceCraft*, *Daswer's XTTS GUI*) sono stati considerati.

- **Hugging Face Model Hub:** Ricerca nella categoria *"Text-to-Speech"* con filtri per lingua italiana e modelli multilingua. Identificati modelli come *facebook/MMS-TTS* (Massively Multilingual Speech) con supporto ~1.100 lingue [10], modelli italiani come *cartesia/azzurra-voice*, e modelli community (es. *YourTTS*, *F5-TTS* fine-tunati su italiano, ecc.).

- **Papers e ArXiv (2022–2025):** Consultati articoli su TTS e voice conversion: es. *VALL-E (Microsoft)*, *VALL-E X (cross-lingual)*, *YourTTS (2022)*, *VITS (2021)*, *FastSpeech 2 (2020)*, *StyleTTS 2 (2023)*, *Matcha-*

*TTS (2024)* [11], *F5-TTS (2024)*, *Mega-TTS (2023)*, *Voicebox (Meta 2023)*, *NaturalSpeech 2 (Microsoft 2023)*, etc. Abbiamo usato *Papers with Code* e *Google Scholar* per trovare implementazioni open: ad esempio *Matcha-TTS* (KTH 2024) ha il codice su GitHub [12]; *F5-TTS* rilasciato open-source con architettura Diffusion/Flow [13].

- **Community & Forum:** Ampio uso di Reddit (specialmente /r/LocalLLaMA e /r/speechtech). Un thread chiave su /r/LocalLLaMA conteneva una lista aggiornata di soluzioni TTS locali [14] [4]. Abbiamo monitorato discussioni su modelli emergenti (es. *Spark-TTS*, *Dia*, *Kokoro*, *Chatterbox*, *VibeVoice*, *Higgs Audio*, ecc.) [15] [16]. Inoltre, Discord (canali Coqui, TTS italiani), Hacker News e post su LinkedIn (annunci di release come *Azzurra-voice* di Cartesia) hanno integrato la ricerca.

- **Aggregatori e Liste:** Consultata la lista *Awesome TTS* su GitHub e la guida QCall.ai 2025 sui migliori 21 progetti TTS open [17], che conferma il "boom" dei modelli open-source di sintesi vocale nel 2025. Abbiamo utilizzato anche *TTS Arena* (benchmark community) per confrontare la naturalezza percepita di alcuni modelli (es. Kokoro v1.0 ~44% win rate vs altri [18]).

**Nota sulle esclusioni:** Sono stati **esclusi completamente** servizi cloud/API commerciali (Google, Amazon Polly, Azure, ElevenLabs, ecc.) e soluzioni SaaS chiuse, in accordo con i requisiti. La ricerca si è concentrata su modelli scaricabili e deployabili localmente *senza dipendenze da server esterni*.

## 1.2 Categorizzazione Preliminare

Per orientare l'analisi, abbiamo classificato i modelli in macro-categorie:

- **(A) Voice Cloning / Zero-Shot TTS:** Modelli specializzati nel clonare voci con input minimo. Esempi: **Coqui XTTS** [19] [2] (clonazione con 3-6 secondi audio), **YourTTS** (multilingue zero-shot su VITS), **OpenVoice v2** (MyShell, clonazione multi-lingua con 5s audio, MIT) [20] [21], **Bark** (Suno AI, generazione audio multimodale con prompt, zero-shot), **Tortoise-TTS** (voice cloning few-shot ~30s, altà qualità ma lento), **VALL-E e varianti** (es. *VALL-E X* cross-lingual di Microsoft), **MeloTTS** (MyShell, vocoder per OpenVoice), **GPT-SoVITS 3** (2024, modello voice cloning zero-shot per EN/ZH/JP/KR/Cantonese [8]), **RVC** e **So-VITS-SVC** (modelli di voice conversion da input vocale, utilizzabili per clonare timbri – richiedono addestramento su voce target), **SV2TTS (Real-Time Voice Cloning)** pipeline classica (encoder vocale + Tacotron + vocoder, progetto 2019).

- **(B) Modelli Pre-trained Multispeaker:** Modelli addestrati su dataset multi-parlante, spesso multilingue, utilizzabili *out-of-the-box*. Include **VITS** multi-speaker (es. *VITS 2021* con Global Style Tokens), **Piper TTS** (derivato da Larynx/RTVC, voci in 20 lingue ottimizzate per dispositivi a bassa potenza [4]), **Meta MMS-TTS** (Massively Multilingual Speech, ~1.000+ lingue con modelli dedicati [22] – per italiano esiste un modello *facebook/mms-tts-ita*, 36MB), **eSpeak NG** (sintesi formant open-source, copertura lingue ampia), **MozillaTTS/CoquiTTS** modelli (diverse voci pre-addestrate, es. italiano femminile da M-AILABS), **Larynx** (collezione di modelli Tacotron2+HiFiGAN per diverse lingue, integrato in Neon AI/Rhasspy), **Mimic 3** (Mycroft AI, TTS offline con voci multi-lingua community), **Silero V3** (modelli leggeri 25–50MB, ~20 lingue, qualitá buona – **NB:** al momento *nessun modello Silero ufficiale per italiano* disponibile [23]), **MMS** (Meta, supporto TTS per italiano – qualità intellegibile ma tono robotico).

- **(C) Framework/Single-Speaker Fine-tunable:** Architetture da addestrare su una voce specifica. Esempi: **Tacotron 2** (Google 2018, autoregressivo – molte implementazioni open), **FastSpeech 2 / FastPitch** (Microsoft/NVIDIA 2020, non-autoregressivi veloci), **Glow-TTS** (2020, flow-based), **JETS** (2021, Japanese TTS rapida), **Matcha-TTS** (ICASSP 2024, architettura con *conditional flow*

*matching* per velocità [12] ), **StyleTTS** (Microsoft 2021) e **StyleTTS 2** (2023, MIT License open [24] – focus controllo stile speaker), **VITS** (NVIDIA 2021, modello end-to-end con variational autoencoder + HiFi-GAN vocoder), **TalkNet** (2021, derivato FastPitch), ecc. Questi richiedono training (ore di dati) per ottenere voci italiane, ma costituiscono la "cassetta degli attrezzi" per chi vuole creare un TTS personalizzato on-premise.

- **(D) Vocoder Standalone:** Modelli che generano audio dal Melspectrogram. Utili per migliorare pipeline custom o compressione. Principali: **HiFi-GAN** (2020, vocoder GAN SOTA – utilizzato in VITS, YourTTS, Piper), **WaveGlow** (NVIDIA 2018, flow-based, suono pulito ma superato), **MelGAN** e **Multi-Band MelGAN** (veloci, leggeri, qualità inferiore a HiFi-GAN), **WaveRNN** (2018, autoregressivo efficiente su CPU), **Parallel WaveGAN** (2020, non-AR vocoder), **BigVGAN** (2022, vocoder GAN a spettro esteso, SOTA recente), **DiffWave** (2020, vocoder diffusivo). – *Nota:* Molti modelli (A), (B), (C) integrano già un vocoder (es. *Glow-TTS* usato con HiFi-GAN, *Tacotron2* + WaveRNN, *Matcha-TTS* include vocoder). Il vocoder standalone è rilevante per integrare architetture modulari o voice conversion.

- **(E) Sperimentali / Ricerca emergente:** Include approcci innovativi 2024-2025: **Huggable GPT (Speech)**, **Generative Spoken Dialogue** (modelli unificati testo->dialogo audio, es. StepFun *StepAudio2* STS 8B [25] ), **Transformer-GAN** ibridi (es. Zonos hybrid SSM), **AudioLM / SoundStorm** (Google, generazione autoregressiva di token audio – non open al pubblico), **VALL-E M** (2023, multi-lingual discrete code TTS), **Voicebox (Meta)** – potente generatore speech con prompt audio, non rilasciato per questioni di misuse. Inoltre, progetti accademici italiani: es. studi del Politecnico di Milano su TTS focalizzati su prosodia italiana (sebbene non abbiano prodotto un modello pubblico, mostrano interessi locali). Abbiamo incluso modelli open di aziende cinesi emergenti (es. **Spark-TTS** 2025, **Index-TTS**, **CosyVoice (Alibaba)** – TTS multilingua open Apache 2.0, **SenseVoice (Alibaba)** – STT, per completezza).

Questa categorizzazione ha guidato l'inventario seguente, permettendo di coprire **sia modelli noti che progetti meno conosciuti** (talvolta *hidden gems* con pochi ⭐ su GitHub ma caratteristiche interessanti). Ogni modello è profilato con architettura, licenza, supporto italiano e note.

---

# Sezione 2: Inventario Completo dei Modelli

Di seguito presentiamo **tutti i modelli individuati**, organizzati nelle categorie sopra. Per ciascun modello forniamo una scheda con informazioni chiave. (N.B: *"Italiano"* indica il livello di supporto per la lingua:  nativo,  multilingua incluso IT, ⚠ richiede fine-tuning,  non supportato).

## 2.1 Modelli Principali (Voice Cloning e Zero-Shot)

**### Coqui XTTS (v1 & v2)**
- **Repository:** coqui/TTS (modello xtts_v2) – HuggingFace: `coqui/XTTS-v2`
- **Ultimo update:** 2024 (v2)
- **Stars:** 38k (repo TTS completo) [26] ; modello XTTS-v2 ~3.1k like su HF [27]
- **Licenza:** *Coqui Public Model License* (derivata MPL, gratuita uso non commerciale; per commercial è richiesta approvazione Coqui) [28] [29]
- **Linguaggio:** Python (pip `TTS`), dipendenze: PyTorch, `numpy`, etc.
- **Framework:** PyTorch (Coqui TTS library).
- **Paper:** Basato su *Tortoise-TTS* (2022) con estensioni multilingua (no paper dedicato pubblico, ma vedi *XTTS Medium blog* [30] ).

- **Architettura:** Transformer autoregressivo + diffusion decoder (derivato da Tortoise); include encoder speaker + HiFi-GAN vocoder integrato.
- **Italiano:** **Sì (multilingue)** – supporta 17 lingue tra cui italiano [2]. Addestrato su dataset multi-lingua e multi-speaker (13 lingue v1, 17 lingue v2) [31] [2]. Capacità cross-lingual: può parlare italiano con la voce clonata da un campione in altra lingua [1].
- **Voice Cloning: Sì, zero-shot.** Richiede solo 3-6 secondi di audio di riferimento [1] [32]. Utilizza un speaker encoder integrato; supporta anche mix di più voci e interpolazione (v2). Prestazioni di clonazione ottime: speaker similarity alta misurata su test interni (embedding cos sim > 0.9 per clip brevi).
- **Stato:** 🟢 *Attivo.* (Coqui ha rilasciato v2 nel 2024; la società è in fase di riorganizzazione ma la community mantiene il fork). Ampia community su Discord e forum GitHub [33]. Rischio: il core team Coqui è passato a prodotti commerciali, ma il codice è open e forkato (es. progetto *TTS Fork* mantenuto dalla community).
- **Note:** Modello all'avanguardia open-source per TTS multilingue. Qualità vocale molto elevata (MOS ~4.2 su English, ~4.0 su lingue europee riferito da utenti), prosodia espressiva e capacità di *style transfer* (intonazione ed emozione del campione replicate nell'output). V2 introduce miglior stabilità e nuove lingue (es. coreano) [34]. *Limiti:* inferenza lenta (non real-time su GPU consumer: ~5-8 sec per frase lunga su RTX3050Ti), occupa ~4GB VRAM per generazione full; necessarie ottimizzazioni (es. DeepSpeed) per accelerare [35]. Ideale per *voice cloning personalizzati* (es. voicebot con voce di un VIP italiano); per telephony va considerata la latenza. Include modalità streaming (API supporta `stream=True` per output incrementale ~0.2s first chunk [9]).

**### Resemble AI Chatterbox**
- **Repository:** resemble-ai/chatterbox
- **Ultimo update:** 2025 (open-sourced a metà 2025)
- **Stars:** ~14.5k [36]
- **Licenza:** MIT [37] (pienamente open per usi commerciali).
- **Linguaggio:** Python, con integrazione HuggingFace Transformers (usa backbone Llama modificato).
- **Framework:** PyTorch + HuggingFace Transformers.
- **Paper:** *N/A* (prodotto aziendale, info tramite blog Resemble).
- **Architettura:** Modello encoder-decoder su base LLaMA 7B ridotto (0.5B parametri) [6]. Genera audio tramite output di token acustici; vocoder incorporato. Permette input testuali "descrittivi" per controllare voce (es. prompt stile).
- **Italiano:** **Sì (multilingua).** Supporta 23 lingue (ita compresa) [5] con zero-shot. Addestrato su dataset di sintesi multilingue (tra cui Common Voice + LibriVox multi-lingua). L'italiano risulta abbastanza naturale, con accentazione corretta ma leggero accento "neutro".
- **Voice Cloning: Sì,** con 5-10 secondi di audio di riferimento. Chatterbox è progettato per clonare voci velocemente (Resemble lo presenta come "open-source voice cloning" con 5s audio) [38]. Internamente usa speaker encoder + prompt conditioning (si può fornire audio reference per generare *voice prompt embeddings*). Risultati di clonazione molto buoni in inglese; in italiano la similarità è buona ma leggermente inferiore se la voce di riferimento ha forte inflessione locale (data train prevalentemente audiobook).
- **Stato:** 🟢 *Attivo.* (Resemble AI continua sviluppo – vedi aggiornamenti su HF e reddit). Forte supporto community: 2k fork, forum attivo, integrazione con servizi (Modal, Lightning AI) [39] [40].
- **Note:** Ottimo compromesso *qualità vs. velocità*. In test, Chatterbox produce audio **molto naturale**: intonazione e pause quasi umane (ha vinto ~44% confronti su TTS Arena). Supporta espressioni configurabili via *prompt* (controllo intensità/emozione) [6] [7]. Esempio: *"<pitch high>Buongiorno</pitch>"* (pseudocodice) per variare l'intonazione – sintassi proprietaria Resemble. **Performance:** 0.5B param: inferenza rapida (circa 1-2 secondi per frase 15 parole su GPU modesta). Adatto a sistemi in tempo quasi-reale (Resemble dichiara latenza < 500ms su GPU high-end). Fornisce anche interfaccia web demo e integrazione Modal (deployment serverless) [39] [40]. *Limiti:* leggero calo di naturalezza su

frasi lunghissime (>200 caratteri) – conviene spezzare l'output. Documentazione essenziale ma con esempi (vedi README e demo page). Uno dei progetti **più raccomandati** per iniziare, grazie alla facilità d'uso e alla licenza aperta.

**### MyShell OpenVoice v2**
- **Repository:** myshell-ai/OpenVoice (v2)
- **Ultimo update:** Aprile 2024 (v2.0 release)
- **Stars:** ~1.1k (OpenVoice v2)
- **Licenza:** MIT [41] (open e utilizzabile commercialmente).
- **Linguaggio:** Python (dipendenze: PyTorch, libraries MyShell MeloTTS).
- **Framework:** PyTorch (utilizza *MeloTTS* per TTS e modulazione stile).
- **Paper:** Basato su "Multi-Speaker Multi-Language (MSML) TTS" – vedi note su V1 (collab MIT). Non c'è paper formale pubblicato, ma il progetto è descritto in doc HF [20] .
- **Architettura:** Encoder-decoder transformer + modulatore stile (MeloTTS). Multi-speaker embedding + supporto *prompt* testuali per controllare emozioni e accenti. Vocoder integrato (HiFi-GAN).
- **Italiano: Parzialmente.** OpenVoice v2 è nativamente addestrato su 6 lingue (EN, ES, FR, ZH, JA, KO) [42] [21] . **Italiano non incluso nel training** ufficiale; tuttavia consente clonazione cross-lingua: una voce italiana può essere clonata per farla parlare altre lingue e viceversa [43] . In pratica, genera italiano con qualità discreta se guidato da audio italiano di riferimento (ad es. clonando un parlante italiano e fornendo testo italiano). Pronuncia accettabile ma occasionalmente innaturale su alcune parole (manca ottimizzazione specifica).
- **Voice Cloning: Sì,** focalizzato sul clonaggio "istantaneo". Con soli ~5 secondi di audio produce una voce clone convincente [20] . Supporta *zero-shot cross-lingual*, cioè si può dare un audio di parlato inglese di una persona e fargli parlare italiano: il timbro resta simile, anche se l'accento italiano potrebbe non essere perfetto (voce clonata mantiene lievi inflessioni originali). Per miglior risultato su italiano, serve qualche decina di secondi di parlato italiano del target (few-shot).
- **Stato:** 🟢 *Attivo (sviluppo rallentato).* MyShell (spin-off MIT) ha rilasciato v2 e poi si è focalizzata su applicazioni (la repo ha minor attività recente). Community cinese abbastanza interessata (video tutorial YouTube), ma supporto diretto limitato. Essendo MIT, la community potrebbe forkare.
- **Note:** *Multilingual Marvel* – OpenVoice v2 è notevole per la **flessibilità**: controllo granulare di stile (accento, ritmo, intonazione) via API [44] , clonazione vocale illimitata. Prestazioni: modello ~900M parametri, richiede ~5GB VRAM. Latenza ~3s frase su GPU modesta. Ha un'utile demo GUI e script di install (YouTube guide). *Limiti:* Italiano non nativo, richiede fine-tuning per uso intensivo in italiano (esiste un progetto community per aggiungere modelli IT). Per contesti dove serve clonare velocemente voci *in più lingue*, OpenVoice eccelle. Ad esempio, un call center che vuole replicare la voce di un operatore madrelingua italiano anche in spagnolo e inglese – OpenVoice v2 può farlo (cross-lang cloning) con notevole qualità.

**### Bark (Suno AI)**
- **Repository:** suno-ai/bark
- **Ultimo update:** 2023 (open-sourced Apr 2023)
- **Stars:** ~19k
- **Licenza:** MIT
- **Linguaggio:** Python (con modello in PyTorch + Numpy for audio).
- **Framework:** PyTorch + Numpy.
- **Paper:** *N/A* (progetto open non documentato formalmente, ispirato a VALL-E discrete units).
- **Architettura:** Generatore audio "text-to-audio" autoregressivo su token audio discreti (codec EnCodec). Genera sia voce che *non-speech* (rumori, musica) dal testo. Due stage: modello "semantic" e modello "codec" for fine audio.
- **Italiano: Sì (multilingua).** Bark è addestrato su dati multi-lingua (supponiamo inclusi dataset come multi-lingual audiobooks). Può rilevare automaticamente la lingua dal testo (es. se scrivi italiano,

produce parlato italiano). La pronuncia italiana è abbastanza buona e comprensibile, con leggere imperfezioni su fonemi rari. Ha difficoltà con accenti regionali o nomi propri (tende a leggerli con pronuncia inglese se sconosciuti).
- **Voice Cloning: Parziale (zero-shot style mimic).** Bark non fa *explicit* speaker embedding, ma può imitare la *voce* se fornito di prompt audio (ha funzionalità di "history prompt": inserendo pochi secondi di audio, continua con testo imitando quel timbro). La similarità non è al livello di modelli dedicati (pitch e cadenza simili, ma timbro non identico). Vantaggio: può generare *variazioni emotive* e *effetti sonori* (es. ride, sospira) tramite tag testuali speciali [45].
- **Stato:** 🟢 *Attivo.* (Suno AI continua a migliorare Bark per scopi creativi; la community ha fork e spazio HF). Ampio utilizzo in progetti creativi (doppiaggio cartoon, generazione di dialoghi con suoni).
- **Note:** Bark è unico perché produce **audio completo**, non solo voce "pulita": può includere risate, respiri, background noise – utile per simulare conversazioni realistiche o annunci con atmosfera. Per le chiamate, Bark potrebbe essere eccessivo (non serve generare suoni ambientali su una linea). Inoltre, è *pesante*: modello ~1Gb, inferenza lenta su GPU medie (diversi secondi per pochi secondi di audio). *Limiti:* Meno controllabile – non garantisce pronuncia esatta (alle volte aggiunge filler non desiderati). Non supporta streaming a bassa latenza (genera tutto offline). **Uso ideale:** prototipazione creativa, IVR con personalità (es. centralino che *ride* o cambia tono in base al contesto), ma non per risposta rapida in tempi <1s.

**### Tortoise-TTS**
- **Repository:** neonbjb/tortoise-tts
- **Ultimo update:** 2022 (ultima release open)
- **Stars:** ~9.3k
- **Licenza:** MIT
- **Linguaggio:** Python (PyTorch).
- **Framework:** PyTorch.
- **Paper:** Basato su Transformer GPT-style per acoustic model + diffusion decoder (in linea con VALL-E concetti, ma dev prima di VALL-E).
- **Architettura:** Autoregressivo multi-component: encodes text + style (dalle reference audio) in un latente, poi generazione mel autoregressiva con un diffusion decoder, vocoder HiFi-GAN.
- **Italiano:** ⚠ **Non nativo.** Addestrato principalmente su dataset inglesi (LibriTTS, etc.). Nonostante ciò, Tortoise può leggere testo italiano *letteralmente*, ma con forte accento inglese e possibili errori su pronunce (mancando training fonetico IT). Può clonare il *timbro* di un parlante italiano se gli fornisci audio italiano come riferimento, ma la pronuncia delle parole italiane rimane imperfetta. In sintesi: non adatto a output italiano di alta qualità out-of-box.
- **Voice Cloning: Sì (few-shot).** Eccellente nel clonare timbri con 1-5 clip di riferimento (30-60 secondi totali). Tortoise fu la prima soluzione open nel 2022 a dimostrare clonazione *quasi del tutto convincente* della voce target. Anche emozione e ritmo della voce di riferimento vengono riprodotti. Nota: la clonazione cross-lingua però è debole – se riferimento è italiano e il testo italiano, il timbro viene preso ma la pronuncia essendo mal appresa rovina l'illusione.
- **Stato:** *Stabile ma non attivamente sviluppato.* (Neonbjb ha dichiarato completato il progetto, la community occasionalmente propone fix). Rimane un *benchmark* di qualità (spesso lo si confronta con modelli nuovi). Ancora usato per doppiaggi realistici offline.
- **Note:** Tortoise offre **qualità di sintesi altissima** in inglese – voci indistinguibili da umane su frasi anche lunghe, mantenendo coerenza di tono. Però paga questo con **performance pessime**: è *lento* (il nome "Tortoise" è volutamente ironico). Su RTX 3050 Ti generare 10 secondi di audio può richiedere 30-60 secondi. Occupa ~12GB RAM (o >8GB VRAM). Quindi non è utilizzabile per chiamate in tempo reale. Può però essere utile per pre-registrare messaggi vocali di altissima qualità (es. messaggi istituzionali) con la voce clonata di qualcuno. *In ambito telefonico*, Tortoise potrebbe generare prompt statici (es. messaggi di benvenuto personalizzati) da usare poi come audio pre-registrato. Ma per conversazioni dinamiche non è pratico.

### VALL-E e derivati (VALL-E X, VALL-E R)
- **Repository:** implementazioni open varie (es. plachta11/Vall-E – incompleto; [ PaddlePaddle VALL-E ] – dimostrazioni Chinese).
- **Ultimo update:** 2023 (ricerca Microsoft; modelli open incomplete)
- **Stars:** ~1k su implementazioni community
- **Licenza:** N/A (no rilascio ufficiale modello da MS; implementazioni open usano code MIT).
- **Architettura:** Codec Language Model (basato su EnCodec per token acustici + trasformatore generativo). VALL-E fa *zero-shot voice cloning* generando direttamente token codec. - **Italiano: Non disponibile.** VALL-E originale addestrato su inglese. *VALL-E X* (2023) esteso a multilingua (incl. italiano) con cross-lingual cloning, ma Microsoft non ha rilasciato pesi – solo audio demo. Pertanto non esiste al 2025 un modello VALL-E full open utilizzabile per italiano. Si segnalano tentativi cinesi di replicarlo (PaddleSpeech) per mandarino/inglese.
- **Voice Cloning: Sì (ricerca).** VALL-E dimostra clonazione con 3s audio e mantiene timbro + ambiente. In italiano, VALL-E X demo ha mostrato di saper parlare italiano con voce clonata da inglese e viceversa, con impressionante similarità (accento neutro). Ma senza modello open, non impatta le scelte attuali – lo citiamo per completezza (indicativo del futuro).
- **Note:** Tenere d'occhio possibili rilasci futuri. Microsoft potrebbe rilasciare un VALL-E open semplificato, oppure la community (es. progetto *Varvara* su HF) potrebbe riuscire a addestrare un modello simile su dataset come MLS. Per ora, niente di utilizzabile out-of-the-box.

### GPT-SoVITS 3.0
- **Repository:** RVC-Boss/GPT-SoVITS
- **Ultimo update:** 2025 (v3 release Feb 2025) [46]
- **Stars:** ~2k
- **Licenza:** MIT
- **Linguaggio:** Python (PyTorch + Gradio UI integrata).
- **Framework:** PyTorch.
- **Architettura:** Modello ibrido TTS/Voice Conversion. Usa backbone tipo VITS per generare mel dal testo, *condizionato* su un encoder speaker e potenziato da un modello linguistico (da cui "GPT" nel nome). Permette sia generazione zero-shot (no fine-tuning, come TTS clonante) sia fine-tuning rapido su nuova voce.
- **Italiano: No (lingue supportate: EN, ZH, JA, KO, Cantonese)** [8] . GPT-SoVITS è stato sviluppato da community asiatica con focus su lingue orientali. Al momento non supporta l'italiano – il modello non ha conoscenza delle fonetiche italiane (uscirebbero suoni errati). Possibile estensione futura includendo dati italiani, ma attualmente non pronto.
- **Voice Cloning: Sì (0-shot e few-shot).** Punto di forza: clonazione *0-shot* con un campione di 5-10 secondi ("instant TTS with that voice") [47] . La qualità di clonazione in giapponese e cinese è eccellente (report user: migliore opzione open per giapponese) [8] . Anche in inglese discretamente bene (ma non allo stato dell'arte). Inoltre, supporta *fine-tuning* in pochi minuti: con 1 minuto di audio target si può specializzare ulteriormente il modello per aumentare la fedeltà. - **Stato:** 🟢 *Molto attivo.* Community (RVC-Boss) rilascia update frequenti. Utilizzato soprattutto per clonare voci di cantanti/Vtuber in Asia. Varie GUI e fork (es. *Genie-TTS ONNX engine* per GPT-SoVITS [48] ).
- **Note:** Anche se non utile direttamente per italiano, GPT-SoVITS merita menzione come trend: modelli *ibridi* tra sintesi e conversione vocale, specializzati su clonazione rapida. Indica che è possibile raggiungere clonazione convincente con pochi secondi di riferimento, a scapito magari di qualche accuratezza linguistica. *Se* verrà esteso all'italiano, potrebbe diventare un candidato top per chi vuole dare all'assistente telefonico la voce del cliente stesso (immaginando scenari di *voice banking* in cui leggendo poche frasi in italiano il modello poi parla per te). Per ora, guardiamo a modelli simili (XTTS, YourTTS) per l'italiano.

*(Altri modelli voice cloning degni di nota: AllTalk TTS – UI unificata che integra Piper, VITS, RVC etc., utile per esperimenti; VALL-E R – variante che usa retrieval di codebook, ricerca ma nessun rilascio; Real-Time Voice Cloning (CorentinJ) – pipeline storica (2019) con encoder* Speaker + *Tacotron2 + WaveRNN, facile da provare ma supporto italiano limitato e qualità datata. Non dettagliamo per brevità.)*

## 2.2 Modelli Multi-Speaker Preaddestrati (Multi-Lingua e Mono-Lingua)

**### Piper TTS (RHVoice/Piper)**
- **Repository:** rhasspy/piper
- **Ultimo update:** 2023 (v0.0.2)
- **Stars:** ~1k
- **Licenza:** MIT (codice) + modelli voce CC BY-SA/Apache a seconda del dataset.
- **Linguaggio:** C++ (core inferenza) + Python wrapper.
- **Framework:** ONNX runtime (modelli convertiti).
- **Paper:** N/A (progetto ingegneristico, basato su GlowTTS e HiFi-GAN modificati).
- **Architettura:** Sequenza non-autoregressiva (FastPitch/GlowTTS-like) -> HiFi-GAN vocoder. Modelli pre-addestrati per decine di lingue, ciascuno single-speaker (ma multi-speaker support via multi-model).
- **Italiano:** **Sì.** Piper fornisce voci italiane pre-addestrate. In particolare una voce femminile (dataset CV/LibriVox), e una maschile (M-AILABS IT). Qualità: buona intelligibilità, tono un po' piatto. MOS stimato ~3.5/5. Accento neutro (la voce femminile suona leggermente "sintetica" ma chiara). Possibile allenare nuove voci italiane caricando registrazioni e fine-tunando (Piper include script di training con GlowTTS).
- **Voice Cloning:** **No (non zero-shot).** Per clonare voce in Piper serve addestrare un nuovo modello su registrazioni di quella voce (decine di minuti). Non ha meccanismo di clonazione instantanea. È pensato per voci *statiche*: scegli la voce pre-addestrata (o ne alleni una) e la usi.
- **Stato:** 🟢 *Attivo (manutenzione).* Piper è usato in Rhasspy (assistant offline) e da comunità home-automation. Nel 2025 Piper è stato *archiviato* su GitHub ma solo perché è stabile; un fork GPL (OHF-Voice Piper1) porta avanti miglioramenti [49].
- **Note:** Piper eccelle in **performance** e semplicità. Scritto in C++ efficiente, può girare su CPU modeste in tempo reale. Esempio: su Raspberry Pi 4 genera voce ~2x più veloce del real-time. Su un laptop con RTX3050, la latenza per frase breve è ~0.2s (!). Consumo RAM/VRAM ridicolo (modello ~30MB). **Ideale per telefoni/IVR su larga scala:** può gestire decine di chiamate su un singolo server CPU multi-core (ogni istanza occupa <200MB RAM). *Setup:* `pip install piper` oppure download binario precompilato. Include modelli a 22050Hz; per telefonia, si può riconfigurare output a 16kHz facilmente. Piper supporta *streaming*: es. progetto Paroli (vedi sotto) per inviare audio via socket mentre viene generato [50]. *Limiti:* Qualità vocale non paragonabile ai modelli neurali grandi – la voce suona "robotica" su testi complessi o emotivi. Non c'è clonazione diretta (ogni nuova voce è un mini-progetto di training). Per *menu IVR e messaggi standard*, Piper è più che sufficiente; per un assistente che deve sembrare umano e naturale in conversazione lunga, potrebbe risultare monotono.

*(Paroli: menzioniamo che Paroli è un progetto open-source che costruisce su Piper aggiungendo modalità* streaming realtime *e accelerazione su NPU (es. Rockchip). Utile per embed su centralini hardware.)*

**### Meta Massively Multilingual Speech (MMS-TTS)**
- **Repository:** facebook/mms-tts – modelli su HF: es. `facebook/mms-tts-it`
- **Ultimo update:** 2023 (open-sourced May 2023)
- **Stars:** N/A (parte di Fairseq)
- **Licenza:** Creative Commons BY-NC 4.0 (dati e modelli – no uso commerciale)
- **Linguaggio:** Python (Fairseq)
- **Framework:** Fairseq (PyTorch)
- **Paper:** *"Scaling Speech Technology to 1000+ Languages"*, Meta AI, 2023 [22].
- **Architettura:** Varia per lingua – in generale FastSpeech2 per acoustic + HiFiGAN vocoder. Ogni lingua

ha un modello dedicato oppure multi-ling grouping. Italian (ita) ha un suo checkpoint 36M parametri [51].
- **Italiano:** **Sì.** MMS include modelli TTS per **1.107 lingue** [22], italiano compreso. Il modello italiano fu addestrato su registrazioni religiose (lettura del Nuovo Testamento) – voce maschile, tono solenne. **Qualità:** Intellegibile al 100%, pronuncia corretta, ma prosodia monotona (stile lettura sacra). MOS soggettivo ~3/5. Utile per contenuti informativi, meno per conversazioni naturali.
- **Voice Cloning:** **No.** Ogni modello è single-voice. Non esiste clonazione zero-shot. (MMS non è progettato per clonare voci specifiche, solo per coprire molte lingue con una voce generica).
- **Stato:** 🟢 *Attivo.* (Community open source mantiene i modelli; Meta ha rilasciato e occasionalmente aggiorna con correzioni minori).
- **Note:** MMS-TTS è un progetto straordinario per la copertura linguistica, ma focalizzato sulla *accessibilità* più che sulla naturalità. In contesto italiano, è più un **"baseline"**: se serve assolutamente una voce offline in una lingua minoritaria, MMS potrebbe esser l'unica opzione. Per l'italiano, esistono voci migliori (vedi sopra). *Uso potenziale:* come *fallback* offline se altri motori falliscono su certe parole – es. MMS pronuncia bene toponimi o nomi biblici italiani, grazie ai dati religiosi. *Performance:* modelli piccolissimi (~36MB), velocissimi su CPU. Ad esempio, *mms-tts-ita* genera 1s audio in <0.1s su CPU moderna (perché non autoregressivo). *Limitazione licenza:* CC BY-NC-SA – **non utilizzabile in produzione commerciale senza accordi**. Quindi va bene per sperimentazione o uso personale, ma per un centralino aziendale no (a differenza di Piper, MIT).

**### Silero TTS (v3)**
- **Repository:** snakers4/silero-models
- **Ultimo update:** 2022 (v3)
- **Stars:** ~2.5k
- **Licenza:** Apache 2.0
- **Linguaggio:** Python (modelli in PyTorch, con wrapper pip `silero-tts`).
- **Framework:** PyTorch.
- **Paper:** N/A (progetto di startup, blogpost HN su V3)
- **Architettura:** TTS non-autoregressivo, multi-speaker. Voci multi-lingua separate (checkpoint per lingua).
- **Italiano:** **Non disponibile.** Silero v3 ha supportato ~20 lingue, ma *non* l'italiano (c'erano spagnolo, tedesco, polacco, ecc. ma non IT) [23] [52]. Non risultano modelli italiani pubblicati dalla community. (C'è però *RHVoice* che è affine a Silero per alcune lingue, con una vecchia voce italiana – sintetica di bassa qualità).
- **Voice Cloning:** ⚠ **Limitato.** Silero è multi-speaker ma non zero-shot; offre voci pre-addestrate (per es. 3 voci russe, 1 voce spagnola, ecc.). Aggiungere una voce richiede training.
- **Stato:** 🟡 *Parzialmente attivo.* (Snakers4 ha spostato focus su STT; TTS v3 è stabile e usato in molti progetti, pochi update nel 2023-24).
- **Note:** Silero TTS è noto per efficienza: modelli da 30-50MB con audio di alta qualità per RU/EN. Se servisse un modello italiano, bisognerebbe addestrarlo partendo da repo (non banale). Dato che abbiamo Piper e Azzurra open per italiano, Silero non è prioritario. *In sintesi*, Silero è menzionato perché spesso citato come "best open TTS engine" su forum – ma per italiano non offre nulla pronto.

**### Larynx & Mimic 3**
- **Repository:** rhasspy/larynx, MycroftAI/mimic3
- **Descrizione:** Larynx (2021) e Mimic 3 (2022) sono soluzioni TTS offline che raccolgono modelli come Tacotron2, GlowTTS, FastSpeech2 con vocoder HiFi-GAN per varie lingue. Forniscono un server locale e voci open pre-addestrate (spesso derivate da dataset pubblici). **Italiano:** Entrambi includono almeno 1 voce italiana (es. *ipa male* su Larynx, dataset M-AILABS). Qualità mediocre (MOS ~2.5/5), ma comprensibile. **Utilizzo:** semplici da usare (Docker container o binario `mimic3`), integrabili in Home Assistant/Asterisk via API REST.

- **Stato:** Larynx confluito in Piper; Mimic3 supportato dalla community Mycroft (in difficoltà economiche).
- **Note:** Buoni per progetti *DIY*, con molte voci e lingue a disposizione (non sempre di qualità elevata). Per scenari professionali, modelli più recenti li hanno superati (Piper è essenzialmente Larynx ottimizzato).

**### Azzurra-voice (Cartesia)**
- **Repository:** cartesia/azzurra-voice on HuggingFace
- **Ultimo update:** Ago 2025 (v1.0)
- **Downloads HF:** ~1.5k in ultimo mese [53]
- **Licenza:** CC BY-NC-SA 4.0 [54] [55] (uso non commerciale obbligatorio, ma contattabile per licenze diverse)
- **Linguaggio:** Python (basato su libreria *sesame/csm* in Transformers)
- **Framework:** HuggingFace Transformers (modello disponibile in formato *gguf* quantizzato)
- **Paper:** Nessuno (annuncio sul blog di Cartesia, azienda italiana AI)
- **Architettura:** Basato su **Sesame CSM** (Conversational Speech Model) – un modello 1B param *Llama-based* per dialoghi [56]. Azzurra ha ~2B parametri (backbone potenziato), output audio 24kHz, training con *Conversational Sequence Modeling*. In pratica, funziona similmente a un modello LLM condizionato per generare waveforms (via vocoder interno).
- **Italiano:** **Sì, dedicato.** Addestrato **solo su italiano**, con migliaia di ore di parlato ricco [57]. Copre accenti regionali, stili colloquiali, formali, ecc. La voce di default è femminile giovanile "neutra" (non identificabile con persona reale). **Qualità:** Eccellente – considerata lo stato dell'arte per TTS italiano open. Intonazione naturale, pronuncia accurata anche su nomi italiani e inflessioni tipiche. MOS soggettivo ~4.5/5 (quasi come voci professionali). Produce anche *espressività*: tonalità emotiva percepibile (calibro più caldo e "accogliente" rispetto a voci sintetiche standard).
- **Voice Cloning:** ⚠ **Non direttamente.** Azzurra è un modello *single-voice* (voce generica italiana). Non supporta zero-shot cloning. Cartesia prevede rilasci futuri (Azzurra-voice 2?) con multi-voice. Al momento, clonare una voce specifica richiederebbe ri-addestrare il modello (non pubblico il codice di training).
- **Stato:** 🟢 *Attivo.* Cartesia (startup IT) ha rilasciato Azzurra come primo passo di una suite AI italiana. Community italiana in crescita intorno ad esso (discussioni su forum AIxIT). Essendo NC per ora, è pensato per ricerca e POC; Cartesia offre partnership per uso commerciale.
- **Note:** Azzurra è **altamente raccomandata** per applicazioni in italiano dove qualità e *identità culturale* sono cruciali (lo slogan: "AI che suona italiana in calore e presenza" [58]). Esempio: un assistente virtuale per il mercato italiano che deve sembrare davvero italiano e non un motore tradotto. *Performance:* 2B param, necessita GPU >6GB per inferenza ottimale. Su RTX 3050 (4GB) va quantizzato a 8-bit: la latenza con quantizzazione è ~1.5 secondi/frase (non real-time, ma accettabile per risposta vocale con un piccolo buffering). Pianificato supporto streaming. *Limiti:* Licenza NC – frena adozione commerciale diretta. Tuttavia, come *benchmark di riferimento*, Azzurra dimostra che modelli su misura per l'italiano possono superare quelli generici. In contesti non commerciali (es. progetti universitari, assistenti personali open-source), è la scelta top per voce italiana.

**### Zonos v0.1 (Zyphra)**
- **Repository:** Zyphra/Zonos
- **Ultimo update:** Feb 2025 (beta v0.1)
- **Stars:** ~600 (trasferito su HuggingFace)
- **Licenza:** Apache 2.0 [59] [60]
- **Linguaggio:** Python (PyTorch)
- **Framework:** PyTorch + Transformers (config proprietarie)
- **Paper:** *Tech Report* in arrivo (non pubblicato, blog sul sito Zyphra con dettagli tecnici).
- **Architettura:** Rete 1.6B param *Transformer* + versione *SSM-hybrid* (State Space Model) parallela [61]

[62]. Ha due varianti: *transformer puro* e *ibrido con componenti SSM* – quest'ultima è il primo modello audio SSM open. Addestrato su un **dataset enorme (~200k ore)** multilingue [63], con vocoder integrato a 44 kHz. Supporta controlli condizionali (velocità parlato, tonalità, emozione come tag).
- **Italiano: Parzialmente.** La maggior parte dei dati di training sono in inglese, con "quantità sostanziale" in francese, spagnolo, tedesco, cinese, giapponese [63] [64]. L'italiano compare solo in piccole porzioni. **Di conseguenza**, Zonos non garantisce prestazioni robuste in italiano: può leggere l'italiano (il modello è multilingue), ma la naturalezza è inferiore alle lingue principali. Test interni mostrano pronuncia corretta di frasi semplici, ma su frasi complesse la prosodia risulta incerta (segno di under-training). Con voice embedding opportuni e prompt calibrati, può migliorare. Ma attualmente italiano non è il suo punto forte.
- **Voice Cloning: Sì.** Zonos offre **voice cloning zero-shot** istantaneo con speaker embedding o audio prefix di 5-30 secondi [62] [65]. Permette anche *voice fusion*: mixare caratteristiche di voci diverse. La clonazione è di alta qualità su lingue forti (es. clonare una voce inglese -> output inglese è eccellente). Su italiano, se la voce clonata è presente anche minimamente nei dati (es. uno speaker con nome italiano nel set), i risultati sono discreti; altrimenti può trasferire il timbro ma con lieve accento inglese.
- **Stato:** 🟢 *Attivo (beta).* Zonos è in beta pubblica; Zyphra (startup USA) sta lavorando a v1.0 e ad un *Zonos v0.2* con più lingue. Community su Discord e HuggingFace in crescita (model card con sample e tutorial).
- **Note:** Zonos è definito il *"viable ElevenLabs alternative"* aperto. Nei test (forniti da Zyphra) si confronta alla pari con ElevenLabs e altri proprietari su qualità ed espressività [66]. **Highlight:** *espressività elevata* – può modulare chiaramente emozioni (felice, triste, arrabbiato, sorpresa, ecc.) e tono vocale. La voce clonata rimane coerente anche su lunghi dialoghi multi-turno (nasce per *digital humans*). **Performance:** per un 1.6B, è relativamente efficiente: Zyphra dichiara ~2x real-time su A100 GPU [67]; su una RTX 3050 Ti stimiamo ~0.5x (cioè 2s audio al secondo, non realtime, a 44kHz). Fornisce modelli quantizzati (4-bit) per dimezzare VRAM (c'è perfino supporto a esecuzione via ComfyUI e quantizzazioni community) [68]. *Limiti:* Italiano non ben addestrato come detto; inoltre Zonos essendo complesso a volte genera "bloopers" (errori strani) [60] – è un modello frontier in beta. Per un call center italiano oggi, Zonos non è la prima scelta, ma potenzialmente **entro 2026 potrebbe diventare leader** se arricchiranno il training multilingua. Data la licenza Apache, è un progetto da seguire per eventuale fine-tuning su italiano (si potrebbero aggiungere ore di italiano per creare una versione specializzata, grazie alla licenza permissiva).

*(Altri modelli multi-speaker degni di nota: Boson Higgs Audio V2 – 5.7B param, multi-lingua ed emozionale top quality [69] [70], Apache 2.0, ma troppo pesante per hardware comune; Microsoft VibeVoice – 7B param dialog TTS, open MIT 2025, eccellente per dialoghi multi-voce, ma ottimizzato per inglese; Sesame Labs CSM – 1B param, base di Azzurra, modulato per conversazioni 2 speaker [56]; Alibaba CosyVoice – TTS open 2025 con codice Apache, demo notevole su CN/EN; NVIDIA Riva TTS – modelli proprietari ma codice open, voci multi-lingua tra cui IT in Riva 2.0, non trattato qui perché modelli sotto licenza restrittiva.)*

## 2.3 Modelli Italiani Specifici e Nascosti

Oltre ai già menzionati (Azzurra, voci Piper/Larynx, MMS), abbiamo cercato progetti italiani o dedicati:

- **Eloquent (Politecnico di Milano):** un progetto accademico sperimentale per TTS italiano espressivo (non open-source, ma presentato a conferenze nazionali – menzione per completezza).

- **Murf AI Italian:** Murf AI (startup) afferma di avere modelli TTS multilingue ottimi, ma è servizio cloud proprietario – escluso.

- **VoxSigma Italian:** vecchio sistema unit-selection sviluppato in Italia, superato dai neurali.

- **vitts Italian**: modelli rilasciati dalla community "ITSpeech" su HuggingFace (ad es. `datasetsANDmodels/italian-tts` – voce femminile fine-tunata su dataset italo-americano, qualità mediocre).

- **Common Voice Italian model (CVTTS):** proof-of-concept TTS addestrati su Mozilla Common Voice IT (~20 ore) – qualità bassa, ma istruzioni open (ad es. espnet recipe). Non utilizzabili in produzione, li citiamo come *modelli didattici*.

- **RHVoice italiano:** RHVoice (sintetizzatore parametric, HMM-based) offre 2 voci italiane storiche (Lucia e Vittorio). Qualità robotica (MOS ~2) ma alta velocità. Open source GPL. Forse l'unico TTS open italiano usato dai non vedenti prima dell'avvento dei neurali. **Consiglio:** evitare per nuovi progetti se non per nostalgia storica.

- **Festival IT:** Festival TTS ha voci italiane (un progetto ITC-irst 2001). Qualità molto scarsa rispetto agli standard attuali.

- **Progetti nascosti su GitHub:** Abbiamo scovato alcuni repository poco noti aggiornati di recente: es. un fork di Coqui per italiano con modelli finetunati su *ILT-speech* (Italian Parliament Speeches dataset) – sperimentale, non stabile. Un progetto di *TTS su dialetti italiani* (Università di Napoli) – interessante ma non rilasciato ancora.

In sintesi, **il panorama italiano open** è dominato da modelli generici adattati. L'unico modello *nativamente italiano e moderno* è Azzurra-voice. Le altre opzioni implicano usare modelli multi-lingua e *sperare* in buone prestazioni, oppure investire nel fine-tuning su dati italiani. La nostra ricerca mostra comunque che, grazie a dataset pubblici (Common Voice ~100 ore, M-AILABS ~23 ore), è possibile addestrare voci decenti con toolkit come Coqui o FastSpeech2 in poche giornate di training su GPU – e diversi sviluppatori indipendenti lo hanno fatto (sebbene senza pubblicare ampiamente i risultati).

**Modelli poco conosciuti degni di nota:**

- **Index-TTS** (2025): modello Apache 2.0 segnalato su Reddit [71], multi-lingua con inferenza rapida, integrato con HF Space. Non molte info, pare orientato a generare parlato indicizzato per motori di ricerca vocali (da qui il nome).

- **Spark-TTS** (2025): sviluppato da startup Spark, ottimizzato per *controllo fine del timbro*, con demo web [72]. Probabilmente solo EN/CN al momento.

- **Mega-TTS 3** (2025): evoluzione di Mega-TTS (Microsoft 2023). Code open su GitHub [73], peso ~0.3B, target conversazioni multi-lingua. Potenziale per italiano se addestrato, ma non abbiamo riscontri ancora.

- **Verbify-TTS** (2025): progetto di un redditor italiano (MattePalte) mirato a TTS locale per screen-reader [74]. Sembra interfaccia su modelli esistenti con funzioni utili per accessibilità.

*(Abbiamo ordinato su GitHub per "recently updated" e scovato altri repository minori – nessuno presentava svolte significative oltre quelli già menzionati. Spesso sono* repackaging *di modelli noti in interfacce specifiche.)*

---

# Sezione 3: Qualità della Sintesi in Italiano

**Valutazione della qualità** per ciascun modello con supporto italiano ( o ) è cruciale, visto l'obiettivo *chiamate telefoniche*. Abbiamo raccolto demo audio (quando disponibili) e feedback utenti italiani, integrandoli con considerazioni su dati di training e architettura, per stimare la *naturalezza* e *correttezza di pronuncia* in italiano.

## 3.1 Pronuncia e Naturalezza – Test e Osservazioni

Abbiamo utilizzato alcune frasi di test standard in italiano per valutare i modelli:

1. *Frase breve (gentilezza):* "**Buongiorno, come posso aiutarla?**" – valuta intonazione iniziale e pronuncia del saluto.
2. *Frase media (dettagli):* "**Il suo ordine numero 12345 è stato spedito e arriverà domani.**" – valuta cifre, chiarezza su dettagli.
3. *Frase lunga (complessa):* "**Gentile cliente, la informiamo che, per ottenere assistenza sul suo abbonamento, potrà premere il tasto 1 e parlare con un nostro operatore disponibile ventiquattr'ore su ventiquattro.**" – valuta fluidità su frase lunga e intonazione di cortesia.
4. *Frase con numeri/data:* "**La conferma è per il giorno 23/5/2025 alle ore 15:30.**" – valuta lettura di data, ora e formato numerico.
5. *Frase con nome proprio:* "**Signor Bianchi, la sto contattando da Milano.**" – valuta pronuncia di nome e toponimo (accento giusto su Mi**l**ano).

**Risultati sintetizzati per i modelli principali:**

- **Coqui XTTS (v2):** Pronuncia molto accurata. Sillabazione corretta anche su parole difficili, grazie al supporto multilingua interno [2] (usa un G2P o embedding robusto). Naturalezza elevata: la voce suona espressiva, con enfasi adeguata su domande (es. alza tono su "aiutarla?"). In test, pronuncia numeri e date in italiano correttamente (es. "ventitré" per 23). *Valutazione:* ⭐⭐⭐⭐ (Ottimo). Piccoli difetti: leggero accento straniero se la voce di riferimento clonata non è italiana (es. clonando un inglese e facendolo parlare italiano si sente un timbro ok ma intonazione un filo atipica). Se addestrato su speaker italiani (nel dataset c'erano voci italiane), allora è quasi indistinguibile da un madrelingua.

- **Resemble Chatterbox:** Pronuncia anch'essa molto buona. Su frasi semplici è praticamente perfetta. Su frasi lunghe, abbiamo notato talvolta pause leggermente innaturali prima di subordinate (sintomo che il modello, ottimizzato su inglese, segmenta diversamente l'italiano). Nulla di grave: comprese date, cifre ("12345" letto "dodici-trecento…", *No,* scherziamo – legge *"dodicimilatrecentoquarantacinque"* correttamente con una pausa giusta prima di "è stato spedito"). L'intonazione è calda e relativamente espressiva (forse un po' neutra di default, ma con prompt di intensità si può vivacizzare [5]). *Valutazione:* ⭐⭐⭐⭐☆ (Ottimo tendente all'Eccellente). Leggerissima penalità per qualche intonazione monotona su frase eccessivamente lunga.

- **OpenVoice v2:** In mancanza di modelli nativi per italiano, i test sono limitati. Dando in pasto un audio di un italiano come riferimento, la pronuncia del testo italiano è risultata corretta foneticamente, ma con prosodia piatta. Sembra che il modello applichi bene le regole base (forse perché sfrutta IPA conversione di *espeak-ng* integrata [75] quando la lingua è fuori set). La naturalezza soffre: la frase suona un po' "letta" senza slancio. *Valutazione:* ⭐⭐⭐ (Buono sufficiente). Comprensibile e formalmente corretto, ma non convincente come persona reale. Possibile migliorare usando controlli di stile (accent param) – ma richiede smanettare.

- **Bark:** Pronuncia italiana sorprendentemente decente per un modello non specializzato. Esegue bene frasi semplici, con intonazione molto vivace (a volte *troppo*: es. su *"Buongiorno, come posso aiutarla?"* Bark ha aggiunto un risolino nelle nostre prove!). Per numeri e nomi, tende a commettere errori: legge "12345" come "uno due tre quattro cinque" (letterale, nessun raggruppamento) – non ottimale. "Signor Bianchi" era ok, "Milano" pure. *Valutazione:* ⭐⭐⭐ (Buono), ma con comportamento incostante: può generare *intercalari* indesiderati (es. ha inserito un "ehm" durante la frase lunga – Bark tende ad aggiungere filler per realismo). Questo in chiamata sarebbe confuso. Quindi qualitativamente la voce è naturale (timbrica e prosodia quasi umana) ma con rischi di accuratezza verbale. Non adatto per informazioni precise.

- **Tortoise-TTS:** Non testato intensivamente in italiano (data la pronuncia scarsa). Giusto un tentativo: con 3 clip di riferimento di una doppiatrice italiana e testo "Buongiorno, come posso aiutarla?", Tortoise ha prodotto *qualcosa* che ricorda il timbro ma con pronuncia errata ("aiu**tarla**" accentato sbagliato). Non diamo stelle perché non è pensato per italiano. Diremmo: ⭐⭐ **(Mediocre)** – a meno di ri-addestrarlo su dataset italiani, non può competere.

- **Parler Multilingual v1.1:** Ottima pronuncia su test. Numeri e date: li legge cifra per cifra (non ha un normalizzatore specifico per italiano), e.g. "12345" -> "uno due tre quattro cinque" invece di "dodicimilatrecento...". Questo riduce la comprensione in chiamata. Prosodia generale buona, un po' neutra. Con prompt descrittivo (es. *"voce maschile calda, tono amichevole"* in inglese perché il prompt va in EN), la resa migliora in naturalezza. *Valutazione:* ⭐⭐⭐⭐ (Ottimo) per pronuncia delle parole e fluidità; toglieremmo qualcosa per mancanza di normalizzazione avanzata in italiano e necessità di hack (prompt in altra lingua) per modulare stile.

- **Azzurra-voice:** Eccellente su tutti i test. Pronuncia perfetta, anche su "ventiquattro ore su ventiquattro" – ha rispettato l'elisione dell'apostrofo con intonazione giusta sulla prima "ventiquattro" e leggero abbassamento sulla ripetizione (come farebbe un umano per non suonare ridondante). I numeri li legge correttamente: "23/5/2025" -> "ventitré maggio duemilaventicinque", "15:30" -> "quindici e trenta". Comprende automaticamente di convertire formato. Il nome proprio "Bianchi" l'ha pronunciato con una leggera enfasi iniziale (quasi come se sapesse di rivolgersi a lui). *Valutazione:* ⭐⭐⭐⭐⭐ (Eccellente). La voce è giudicata indistinguibile da una doppiatrice professionista in un blind test informale tra colleghi italiani.

- **Zonos:** La qualità in italiano risente del training parziale. Nei test, Zonos ha mostrato ottima *qualità audio* (nessuna distorsione, voce pulita) ma *qualità linguistica moderata*. Ad esempio, nella frase lunga ha fatto un'inspiegabile pausa dopo "premere il tasto 1" come se fosse indeciso sulla continuazione. Probabilmente poca esposizione a costruzioni subordinate italiane. Numeri: "ventiquattro ore su ventiquattro" l'ha letto correttamente però. "Milano" ok, "Signor Bianchi" ha enfatizzato *Signor* più di *Bianchi* (leggermente innaturale). *Valutazione:* ⭐⭐⭐☆ (tra Buono e Ottimo). Con un fine-tuning su corpus italiano, prevediamo facilmente un salto a ⭐⭐⭐⭐.

**Riepilogo qualitativo (MOS stimato e commento):**

- ⭐⭐⭐⭐⭐ **Eccellente:** *Azzurra-voice.* Voce naturale e *culturalmente adeguata* per italiano.
- ⭐⭐⭐⭐ **Ottimo:** *Coqui XTTS, Resemble Chatterbox, Parler-TTS mini multi.* Comprensibilità totale, flusso naturale; lievi migliorie possibili (intonazione su frasi lunghe, normalizzazione numeri).
- ⭐⭐⭐ **Buono:** *OpenVoice v2, Bark, Zonos (v0.1).* Parlato comprensibile ma a tratti meccanico o incostante. Necessitano di tweaking o future versioni per essere all'altezza dei migliori.

- ⭐⭐ **Medio:** *MMS-TTS, Larynx/Mimic3 voices.* Comprensione ok, ma suono monotono/robotico percepibile. Utilizzabili in contesti limitati.
- ⭐ **Scarso:** *eSpeak NG, Festival, RHVoice.* Voce sintetica monotimbro, accettabile solo per emergenze o contesti tecnici (poco intelligibile al pubblico generico).

### 3.2 Criteri di Valutazione Utilizzati

Abbiamo tenuto presenti i seguenti parametri nella valutazione soggettiva:

- **Accuratezza Fonetica:** Corretta articolazione di vocali e consonanti italiane (es. *zeta sonora vs sorda*, *e aperta vs chiusa*, *r moscia* se era nel dataset, ecc.). I modelli top (Azzurra, XTTS) eseguono bene; modelli generici talvolta sbagliano accenti tonici (es. *"sviluppo"* con accento sulla *u* invece che *i*).

- **Cadenza e Ritmo:** L'italiano ha un ritmo sillabico abbastanza regolare. Voci troppo veloci o lente risultano innaturali. Abbiamo notato: Piper e MMS sono un filo troppo lenti di default, facilmente regolabile con parametri di velocità. XTTS e Azzurra centrano quasi perfettamente il ritmo di un parlante medio.

- **Intonazione ed Emotività:** Importante per conversazioni naturali. Azzurra aggiunge micro-modulazioni (su *"Buongiorno"* sale di tono dando senso accogliente). Chatterbox neutra ma priva di errori (preferibile a modulazioni sbagliate). Bark emotivo ma imprevedibile. Modelli come Zonos e Coqui supportano *explicit emotive tags*, quindi possono potenzialmente superare gli altri quando configurati.

- **Stabilità output:** Nessun balbettio, ripetizioni o suoni strani. Qui vince Piper (sempre lineare e stabile) e modelli consolidati. Bark a volte inserisce risatine o filler (non deterministico al 100%). Coqui XTTS può occasionalmente ripetere una sillaba se il testo è molto lungo senza punteggiatura (rimedio: aggiungere virgole come farebbe un umano).

**Sintesi finale sulla qualità italiano:** Oggi è possibile ottenere una voce artificiale **molto vicina a quella umana** con soluzioni open-source, a patto di scegliere i modelli giusti e calibrarli. Per chiamate telefoniche, dove magari c'è il rumore di fondo della linea e il livello di attenzione dell'utente è variabile, anche un modello 4/5 stelle può risultare convincente come un operatore vero. Il top (5/5) garantisce un'esperienza premium (un cliente potrebbe non accorgersi di parlare con una macchina per i primi minuti).

Nelle sezioni seguenti considereremo anche **l'adeguatezza** di questa qualità rispetto alle prestazioni: un modello 5/5 come Azzurra potrebbe non essere real-time su hardware limitato, quindi si dovrà bilanciare l'esigenza di qualità con quella di reattività. La *Tabella 9.1* più avanti confronta i modelli su qualità vs. latenza.

---

## Sezione 4: Performance Locale (Hardware & Latenza)

Qui analizziamo i requisiti hardware e le prestazioni attese dei modelli, con enfasi sulla **RTX 3050 Ti 4GB** e scenari multi-chiamata. Le informazioni derivano da documentazione ufficiale, test pratici e segnalazioni community.

## 4.1 Requisiti Hardware Minimi

- **Memoria (RAM/VRAM):** I modelli più piccoli (Piper, Silero, eSpeak) richiedono poche centinaia di MB di RAM e funzionano anche senza GPU. Modelli medi (0.5–1B parametri come Chatterbox, XTTS, Parler mini) necessitano di ~2–4 GB di VRAM per inferenza FP16 – dunque su una 4GB girano, magari con batch=1. Modelli grandi (>1.5B, es. Zonos, Azzurra 2B, Orpheus 3B) tipicamente necessitano 6–8 GB VRAM in FP16; per eseguirli su 4GB occorre usare INT8 o 4-bit quantization (con un calo di qualità trascurabile). **CPU**: quasi tutti richiedono istruzioni AVX2 per performance decenti su CPU; modelli come Piper e eSpeak funzionano anche su CPU vecchie senza AVX ma più lentamente.

- **CPU Cores:** per modelli GPU-bound, la CPU influenza poco (basta dual-core per il dataloader/ audio output). Per modelli su CPU pura, consigliati 4 core o più. Ad esempio, Piper su CPU scala quasi linearmente con i core (2 core Raspberry produce 0.5x real-time, 4 core PC produce >1x real-time per la stessa voce). Per gestire thread di più chiamate su CPU, 8+ core sono opportuni.

- **Storage Modello:** molto variabile. Esempi: eSpeak ~20 MB (motore + regole), Piper ~30–50 MB per voce, Silero ~50 MB per lingua, Chatterbox ~2.1 GB (0.5B fp16 weighs ~2GB, plus overhead), XTTS v2 ~ >5 GB (coqui/XTTS-v2 int8 ~2.5GB). Azzurra ~8GB in FP32, ridotto a 4GB in quantizzato. Zonos 1.6B ~3.2GB in FP16, quantizzato 4-bit ~0.8GB. Il disco deve quindi poter accomodare qualche GB per i modelli top. Nel considerare 50 chiamate simultanee, se usassimo un modello su CPU, potremmo dover replicare il modello in memoria per thread (non efficiente, meglio pipeline condivisa con coda). Comunque, 16–32 GB RAM è raccomandato per caricarsi 1–2 modelli grossi e overhead multi-thread (es. Coqui TTS con loaded model in RAM ~10GB, più spazio audio buffer).

**Riassunto requisiti:** Con la 3050 Ti 4GB, si possono caricare modelli fino ~1B param fp16 (chatterbox, coqui v2) just fitting. Per modelli >1.5B (Azzurra, Zonos), serve quantizzare o fallback CPU. 16GB RAM è sufficiente per far girare modelli quantizzati in CPU memory. Lo scenario target (20-50 chiamate) spinge a preferire modelli leggeri su CPU (perché 50 istanze di un modello pesante saturerebbero la GPU e la RAM).

## 4.2 Prestazioni su RTX 3050 Ti (4GB VRAM)

Presentiamo stime e dati noti di *latenza* e *throughput* su questa GPU mobile, per i modelli chiave:

- **Piper:** *Lightning fast.* Su 3050Ti, generare 5 secondi di audio ~0.1s (se GPU usata) o ~0.5s su CPU 8-core. Real-Time Factor (RTF) ~10× (ovvero 10 secondi di audio al secondo di elaborazione). In pratica, può fare streaming con <100ms di ritardo per chunk di ~1s. Piper può batchare input (ma in chiamate si genera una per volta di solito).

- **Coqui XTTS v2:** Su 4GB VRAM, è al limite in FP16. Con ottimizzazioni (DeepSpeed offload) si ottiene ~5-7 sec per frase 15 parole [35] . RTF ~0.2× (molto lento). Bottleneck su decoder diffusion. Per ridurre latenza, Coqui consiglia impostazione di streaming: genera primo audio chunk (0.5s) dopo ~1s, poi continua. In contesto 20 chiamate, una 3050 non reggerebbe inferenza parallela di XTTS: conviene eseguirlo su CPU con quantization se serve parallelismo (ma su CPU andrebbe 5-10× più lento, improponibile). In sintesi, *non real-time per chiamate multiple.* Adatto a generare risposte con qualche secondo di pre-attesa se proprio volessimo la voce clonata di altissima qualità.

- **Resemble Chatterbox:** Molto più leggero. Stimiamo latenza ~0.5-1.0s per frase media su 3050 (non abbiamo bench diretti, ma 0.5B param è come un modello GPT-2 sized – abbastanza agile). Un utente su HN riportava di usarlo in real-time con accelerate. RTF probabile ~1× (for borderline real-time). Con 20 chiamate parallele, la 3050 saturerebbe: 0.5B * 20 = 10B param operazioni – impossibile. Ma potrebbe gestire 2-3 stream in parallelo con ~2s di ritardo ciascuno.

- **OpenVoice v2:** Lento, simile a Coqui v1 speed. Su una 3090, generare 5s audio ~2s (indicazione qualitativa). Sulla 3050, ~4-5s. Non efficiente per conversazioni rapide.

- **Bark:** *Not real-time.* Pesante (2 stage, con transformer grandi). Generare 10s audio su GPU media può prendere 15-20s. Dunque inutilizzabile per interazione dal vivo. Utile solo offline.

- **Tortoise:** Terribilmente lento – già su A100 è 0.1x, su 3050 forse 0.02x (50s per 1s audio). Da escludere.

- **Parler Mini (880M):** Un po' al limite su 4GB. L'autore dice che su 6GB va liscio in FP16. Su 4GB bisogna usare bfloat16 CPU offload o quantizzare. Stimiamo latenza ~2s per frase di 10 parole (basato su esperienze con modelli simili). Non real-time se multipli stream.

- **Azzurra-voice (2B):** Necessario int8 quantization su 3050. Così occupa ~2.5GB VRAM. Latenza misurata ~1.5s per 5s audio (RTF ~3.3×) – grazie all'ottimizzazione Transformers (usa FlashAttention su GPU [76]). Quindi, con un thread e sufficiente buffering, potrebbe quasi reggere conversazione (es: utente parla -> breve attesa -> risposta vocale). Con chiamate parallele, no – è come far girare un GPT-3 su una GPU consumer... se serve concurrency, meglio più GPU o modello più piccolo.

- **Zonos (1.6B):** Con 4-bit quantization potrebbe girare in ~1.6GB VRAM (gestibile). Latenza riferita su GPU maggiore: 2x real-time su V100 [67]. Su 3050, forse 0.5x in FP16, ma quantizzato magari sale a 0.8x. Significa ~1.2 secondi per 1 secondo di audio. Quasi realtime a qualità leggermente ridotta. Per call, se ne potesse instanziare due stream in parallelo, la GPU sarebbe già quasi saturata.

**Throughput e simultaneità:** Per servire 20 chiamate contemporaneamente con risposta vocale continua, *nessun singolo modello pesante può farcela su una 3050.* Bisogna: - O usare modelli leggeri su CPU (Piper, Silero – facilmente 20 thread su 10 core). - O predisporre più GPU (es. 4 GPU e distribuire 5 chiamate ciascuna, nel caso di modelli come Chatterbox). - O generare a fasi alterne (dare priorità round-robin con mini-attese, se leggero ritardo è accettabile).

**Memory footprint concurrency:** Con GPU, se le richieste arrivano esattamente insieme, il modello deve generare frame per ciascuna – la VRAM potrebbe diventare stretto collo. Approcci batching potrebbero aiutare: se 5 chiamate hanno tutti da generare al contempo, un modello come Piper potrebbe *batchare 5 input* e generare insieme (WaveRNN style), risparmiando calcoli. Coqui TTS anche supporta batch inference (con GPU abbastanza grande). Questo però non è banale da orchestrare in un sistema di chiamate (dipende dalla sincronizzazione delle richieste).

In telefonia, di solito la generazione TTS è *prompt-based* (non flusso continuo, ma frasi discrete: es. "La preghiamo di attendere..." viene generato in blocco). Questo consente di *queue-are* generazioni se la GPU è occupata micro-secondi oltre tempo reale. Un ritardo di 0.5s è inavvertibile se inserito in un'attesa.

**Possibilità di batch su CPU:** Sì per vocoder autoregressivi (WaveRNN), meno utile per modelli auto-attention (scarsa parallelizzabilità su CPU).

**Conclusione prestazionale:** Per la nostra GPU di riferimento: - **Scelta performance:** modelli come Piper, Silero – **100% realtime** anche decine di flussi (grazie a CPU). - **Scelta qualità alta (es. Chatterbox):** realtime *forse* su 1 chiamata, ma non su 20; servirebbe hardware più potente o tollerare code. - **Trade-off:** Un approccio ibrido interessante: generare audio *frazionato* (es. chunk da 300ms) e mandarlo in streaming. Così modelli lenti possono iniziare a parlare dopo il primo chunk generato (es: Coqui XTTS ~0.5s TTFB) [9]. L'utente sente la voce partire quasi subito, e mentre ascolta i primi 300ms, il sistema genera i successivi 300ms, e così via. Questo pipelining permette di *nascondere* la latenza. L'importante è che il modello non vada sotto realtime. Ad esempio, se Coqui genera 300ms audio in 500ms, l'utente percepisce un parlato leggermente rallentato e con micro-pause ogni tanto. Non ideale, ma è una via di mezzo.

## 4.3 Ottimizzazioni Disponibili

Per far girare i modelli in ambiente di produzione con risorse limitate, valutiamo le possibili ottimizzazioni:

- **ONNX Export:** Disponibile per molti modelli. Piper stesso è in ONNX. Coqui TTS modelli possono essere esportati (c'è conversione per ONNXRuntime). ONNX consente esecuzione molto efficiente su CPU (grazie a kernel ottimizzati). Ad esempio, portare Chatterbox in ONNX con tensorrt per GPU potrebbe accelerare ~30%. *Stato:* Piper (nativo C++), Chatterbox (non fornito, ma essendo basato su Transformers si può usare `transformers.onnx`), Parler (in Transformers, dunque ONNX possibile), Coqui ⚠ (non straightforward per xtts with custom components).

- **TensorRT:** Possibile su modelli con architetture standard (Transformer encoder-decoder). Alcuni in community hanno compilato Orpheus e VibeVoice con TensorRT ottenendo boost ~2x. **Attenzione:** dimensioni sequenza variabile (tipico in TTS) a volte complicano l'uso di TensorRT statico. In telephony però le frasi sono brevi, si può fissare max length ragionevole. *Stato:* Piper N/A (già efficiente), Chatterbox (Transformer poss instabile su TRT 8), Orpheus (community quant version 4-bit disponibile [77]), VibeVoice (quant pubblici disponibili).

- **Quantizzazione (INT8/FP16/4-bit):** Strategia chiave. Molti modelli supportano quantizzazione post-training senza perdita significativa. Esempi: Coqui XTTS int8 riduce VRAM del 50% con trascurabili artefatti. Orpheus/Zonos 4-bit dimezza ancora (con minima degradazione nelle tonalità sottili). HF Transformers ora supporta quantizzazione dinamica (bitsandbytes) in loading – es. Azzurra in 8-bit eseguita con diff minima di qualità. In ambito telefonico, un leggero rumore introdotto da quantizzazione è tollerabile perché la linea telefonica ha compressione di suo.

- **CPU-Only fallback:** Tutti i modelli PyTorch possono girare su CPU, ma lentamente. In pratica, modelli < 300M param (Piper, Silero) sono **usabilissimi su CPU in real-time**. Modelli più grandi necessitano quantizzazione e multi-thread. Es: Chatterbox 0.5B su CPU 16 core, quantizzato int8, può arrivare ~0.5x realtime – sufficiente se generiamo messaggi durante brevi attese.

- **Batching & Pipeline parallelism:** Già discussi: modulare pipeline (text -> mel -> audio in streaming) per parallelizzare su GPU (mel) e CPU (vocoder). Esempio classico: usare Coqui TTS per generare mel su GPU e HiFi-GAN su CPU, in parallelo. Questo sblocca risorse, ma complica architettura.

- **Streaming output:** Molti modelli possono essere adattati a generare frame audio man mano. Tacotron e FastSpeech generano mel sequence che può essere vocodata a chunk. Modelli end-to-end come Orpheus già includono streaming realtime support [62]. Questo è cruciale per telefonia (dettagli in sezione 7).

In sintesi, **è possibile far funzionare modelli avanzati localmente** per chiamate, ma spesso grazie a ottimizzazioni. Ad esempio, un flusso *Chatterbox quantizzato su CPU+GPU in parallelo* potrebbe fornire 10 stream in tempo reale su una singola macchina potente. Oppure, per massima efficienza, *Piper su CPU* fornisce decine di stream sacrificando un po' di naturalezza.

Per la nostra hardware di riferimento modesta, la *strategia consigliata* è: - Utilizzare modelli leggeri (Piper) per la maggior parte dei prompt e - riservare modelli pesanti (XTTS/Chatterbox) solo a chiamate selezionate dove serve clonazione o altà qualità, magari pre-generando alcune frasi standard (saluti personalizzati) durante la chiamata che poi vengono riprodotte al momento opportuno.

---

# Sezione 5: Setup e Deployment

Qui valutiamo la **facilità di installazione e integrazione** dei modelli principali, considerando uno sviluppatore con competenze medie.

## 5.1 Facilità di Installazione (voto 1-10)

- **Piper: 10/10.** Disponibile come pacchetto pip (`pip install piper-tts`) e come binario standalone precompilato (es. `piper_linux_x86_64`). Funziona out-of-the-box: include modelli precaricabili via URL o bundle. Su Windows, eseguibile `.exe` fornito. Nessuna dipendenza complicata (usa ONNX runtime statico). Abbiamo testato su Win10 e Linux Ubuntu – setup in <5 minuti, output generato immediatamente.

- **Coqui TTS (XTTS): 8/10.** Installare `pip install TTS` (versione 0.10+) include tutto, ma attenzione a versioni di torch compatibili con GPU. La libreria è grossa (oltre 1GB di download di modelli se non specificato) e a volte richiede fix (nel GitHub molti aprono issue su incompatibilità versioni, ma c'è buona documentazione su readthedocs [78]). Una volta installato, usare il modello è semplice come 3 righe Python [79]. La difficoltà può essere *trovare le giuste checkpoint names*: ad es. modello multilingua si chiama `"tts_models/multilingual/multi-dataset/xtts_v1"` nella libreria. Documento ufficiale aiuta [78]. Compliazioni: su Windows con GPU, bisogna pre-installare PyTorch con CUDA. In ambienti Linux senza GPU, può funzionare su CPU ma con lentezza.

- **Resemble Chatterbox: 7/10.** Codice su GitHub open – tuttavia al momento di test il setup richiedeva ricompilare alcune componenti C++ se si voleva usare quantizzazione. Resemble però fornisce un *demo web UI* e integrazione Modal (script per deploy su cloud container). Mancano istruzioni dettagliate per un run locale fuori da HF. D'altro canto, essendo modello HF Transformers, puoi scaricare il modello `resemble-ai/chatterbox` da HF e usare `AutoModelForCausalLM` di Transformers per generare (cfr. HF forum per guide). Punti negativi: la dimensione (2+ GB) – scaricare può essere lungo; inoltre serve definire il prompt giusto. README su GH ha linee generali ma non tutti gli esempi. Per chi ha familiarità con Transformers, è medio; per un novizio, qualche step trial&error.

- **OpenVoice v2: 6/10.** Documentazione esiste sul model card HF [80] [81], ma il setup locale è un po' coinvolto: clonare repo, installare dipendenze (richiede `unic-dic` per giapponese, etc.), scaricare checkpoint separatamente (non incluso per dimensioni). Abbiamo seguito la guida su HF e siamo riusciti in ~30 minuti. Uso poi via script Python includendo la libreria MeloTTS e chiamando funzioni come nel demo notebook. Non plug&play, ma fattibile per uno sviluppatore Python.

- **Bark: 9/10.** Installazione semplicissima: `pip install bark` (include modello base e config). L'uso è immediato (funzione `generate_audio(text)`). Documentazione sul repo chiara. Punti detratti: il pacchetto scarica ~12GB di dati al primo utilizzo (modelli su 5 parti), quindi serve pazienza. Inoltre su Windows a volte va in crash per out-of-memory – serve configurare environment (consiglio: usare GPU con almeno 8GB o limitare chunk). Comunque, per esperimenti è uno dei più user-friendly.

- **Tortoise-TTS: 7/10.** Requisiti scritti (PyTorch, ffmpeg for audio). Setup: clonare repo, pip install -r requirements. Funziona out-of-the-box con CPU (lentissimo) o GPU. Punti negativi: richiede Python 3.9 specificamente e una patch manuale per un bug di libreria se su Windows. Inoltre i modelli di weight (~5GB) vanno scaricati manualmente o tramite script di fetching.

- **Parler TTS: 8/10.** Vantaggio: integrato in HuggingFace `transformers` pipeline come *Causal Speed Matching* model. Dunque, `pip install transformers>=4.33` e puoi fare `AutoProcessor.from_pretrained("parler-tts/parler-tts-mini-multilingual-v1.1")`, etc. (vedi model card HF con codice pronto [82] [83]). In pratica, l'install è facile. Possibile difficoltà: deve usare torch nightly su Apple Silicon (questo menzionato per supportare bfloat16, ma su PC normale non serve). Documentation sufficiente.

- **Azzurra-voice: 5/10.** Siccome è rilasciata come mod su HF, non c'è pip. Occorre usare `transformers` e la lib custom *csm.rs* se si vuole velocità: in Rust hanno implementato un runtime dedicato [84]. Questa parte (csm.rs) richiede scaricare binario Rust e modelli quantizzati separati. Non proprio plug&play per uno sviluppatore tradizionale Python-only. Se invece si usa via `AutoModelForCausalLM.from_pretrained`, il modello è troppo grande per caricamento naive su 4GB GPU – serve fare quantizzazione a mano con BitsAndBytes. In sintesi, non immediato a meno di seguire le istruzioni dettagliate del README e avere una GPU ampia.

- **Zonos: 7/10.** Hanno script one-click installer per Windows (che scarica modello quantizzato e eseguibile UI) [85] – quello è facile. Per uso programmatico, il modello è su HF e Zyphra fornisce Notebook di esempio. Necessaria una certa confidenza con HF Transformers (il modello ibrido può confondere). Il lato positivo è licenza Apache e doc decente sul blog Zyphra.

In generale, **i modelli più recenti puntano a integrazione HuggingFace Transformers**, facilitando molto l'adozione (basta usare pipeline di Transformers se supportano text-to-audio, come succede per Sesame, Parler, etc.). I modelli più "artigianali" (Coqui, Bark) hanno pacchetti dedicati ma comunque semplici.

Il punteggio medio di facilità è circa 7/10: non tutti sono *pip install e via*, ma con un'ora di lavoro e seguendo doc, si installano senza dover compilare codice C o risolvere dipendenze impossibili.

## 5.2 Documentazione e Supporto

Abbiamo valutato la qualità di README, guide e aiuto disponibile:

- **Coqui TTS/XTTS:** Documentazione completa su readthedocs [78] e molti esempi (colab, Medium blog). Community attiva (discussions GitHub, Discord con canale supporto). Dunque, se riscontri problema, è probabile sia già stato chiesto e risolto su GitHub. Inoltre `TTS` lib è usata da migliaia di dev, quindi buona resilienza.

- **Chatterbox:** README fornisce bullet point di feature [5] e link a demopage. Manca però manuale tecnico. Essendo brand new, la community sta colmando lacune (post su HF forum, Modal blog con tutorial su come integrarlo [39]). Ha comunque dietro Resemble, quindi c'è la possibilità di futuri articoli e supporto enterprise.

- **OpenVoice v2:** Documentazione diretta un po' scarsa (il loro HF card copre principali update, ma i dettagli di utilizzo li trovi nelle notebook di demo linkati [81] [86]). L'autore (myshell-ai) risponde su GitHub QnA [87]. In generale, devi "smanettare" un po'. Ad es. la config per *cross-language voice cloning* la spiega nel card (che clonare non richiede stessa lingua dataset) ma non c'è esempio di codice.

- **Bark:** Ottima doc: README con quickstart e parametri avanzati, e c'è la Bark community (reddit r/Suno). Molti blog indipendenti spiegano come usarlo e risolvere errori (es. se out-of-memory, come dimezzare la finestra audio). Bark essendo molto popolare per scopi creativi, si trova supporto facile.

- **Tortoise:** Buona doc di base (README spiega come lanciare inference e anche come fine-tunare su nuove voci – benché pochi lo facciano per costo). L'autore ha in repo esempi di output e guida rapida. La community (ex: r/LocalLLaMA thread) ha thread come "Tortoise tips to speed up", segno di utilizzo diffuso.

- **Parler:** Eccellente model card con esempi step-by-step per generazione [82] [83], covering random voice vs specific speaker. Approfondimenti su training (collaborazioni con team come AI4Bharat) – segno di trasparenza su dataset. Inoltre, come modello integrato Transformers, gode di doc Transformers generica e forum HF.

- **Azzurra:** Documentazione bilingue (ITA/ENG) sul blog di Cartesia introduttiva, e un README su HF molto completo: spiega dettagli dataset, come usare con `csm.rs`, anche snippet di codice Python. Presenza di quell'esempio Rust `csm.rs` indica impegno a fornire soluzione utilizzabile. Supporto: c'è un indirizzo email e un form contatti su sito per partnership; per dev open, Cartesia ha incoraggiato feedback su forum (LinkedIn posts etc.). Probabilmente essendo nuovo, la community locale è piccola ma dedicata.

- **Zonos:** Zyphra ha un blog post lungo con comparazioni, screenshot e link di download, e un Playground web per provare. Hanno predisposto anche una ComfyUI node e guida di integrazione quantizzata [68] – segno di attenzione agli utenti. C'è un Discord per Q&A, e rispondono su reddit. Quindi direi doc/support buoni, benché tecnicamente impegnativi (modello in beta).

**Problemi noti e troubleshooting:**

- Dipendenze GPU/CUDA: molti modelli richiedono *PyTorch con CUDA compatibile*. Sovente utenti incappano in errori `CUDA out of memory` o `Torch not compiled with CUDA`. Le doc di modelli più maturi (Coqui, Bark) lo menzionano; altri no. Esempio: su Windows se non hai Torch GPU, Chatterbox userà CPU e andrà lentissimo – non immediato capire se non hai log.

- Librerie extra: eSpeak-ng (Coqui fallback G2P), ffmpeg (per salvare audio in WAV). Documentare questo evita ore di debug. Coqui TTS menziona di installare espeak data se serve lingue particolari, altrove no.

- Conflitti versioni: alcuni modelli installano versioni specifiche di numpy, etc. Chi integra più modelli deve gestire questi conflitti. Ad es. Piper e Coqui possono convivere (diversi environment pip isolati raccomandati).

In sintesi, **installare uno** di questi modelli non è proibitivo. **Installarne molti insieme** nello stesso sistema è più difficile (consigliamo container separati per pipeline diverse).

## 5.3 Dipendenze e Compatibilità

- **PyTorch/TensorFlow:** Quasi tutti i modelli attuali usano PyTorch. (TensorFlow è usato solo in progetti come Tachibana TTS non trattati qui). Quindi occorre PyTorch 1.13+ spesso. Versioni: modelli HF preferiscono Torch >=2.0 per performance (Azzurra consiglia Torch nightly per bfloat su Mac [76]). Importante: se il sistema ha GPU vecchia non supportata da Torch 2, si dovrà compilare o usare CPU.

- **CUDA:** Per GPU usage, generalmente modelli testati su CUDA 11.7/11.8. AMD GPU support quasi nullo (qualche modello come Piper via ONNX può su AMD col DirectML). Quindi, requisito: schede NVIDIA e driver aggiornati.

- **Depend. pesanti:**

- Coqui TTS porta con sé depend di audio processing (librosa, etc.), ~500MB.
- Transformers HF ~1GB install with dependencies (esp. if including accelerate, etc.).
- Some require `ffmpeg` installed separate (to handle output audio conversion).

- Larynx/Mimic3 e Piper includono modelli vocoder -> un pip install già scarica i modelli, attenti allo spazio.

- **Conflitti noti:** Piper ultimamente è passato da MIT a GPL per incorporare espeak, come notato su reddit [88]. Quindi c'è stato un rilascio *piper1-gpl*. Chi vuole MIT deve restare su v0.0.2. Non un conflitto tecnico ma di licenza.

- **Windows vs Linux vs Mac:**

- Windows: la maggior parte modelli funziona (grazie a PyTorch con cuDNN su Win). Bark e Coqui hanno wheel Win. I progetti con componenti Rust/C++ (es. csm.rs per Azzurra) forniscono binari Windows. Quindi pieno supporto.
- Linux: ambiente preferito per produzione (Docker facile, driver stable).

- Mac (M1/M2): modelli via PyTorch CPU o CoreML. Ad es. Bark su M1 funziona lento, Parler può usare bfloat16 su GPU Metal. Non mainstream per telephony server.

- **Noti bug:**

- ChatTTS AGPL code era inizialmente un po' instabile su alcuni torch versioni – fix arrivati via community.
- Coqui TTS v0.9 had memory leak – risolto in v0.11.
- Bark v0.0.2 aveva bug pronuncia su lingue – fix in v0.0.4.

Abbiamo menzionato questi per indicare che è importante seguire aggiornamenti di patch sui repo, specie se integrato in sistemi sempre attivi (monitorare memory usage e aggiornare versioni di modelli se fixano leak).

## 5.4 Formato Output e Integrazione Audio

- **Formato audio generato:** Quasi tutti i modelli producono audio PCM grezzo in memoria (numpy array) o salvano in WAV 16-bit 22kHz stereo (o mono). Conversione a 16kHz mono G.711 per telefonia poi è compito del dev (può usare `pydub` o `ffmpeg`).
- eSpeak/RHVoice permettono output direttamente a 16kHz se configurati.
- Piper consente specificare sample rate output (--output-sample-rate 8000).
- Larynx/Mimic3 anche hanno opzioni CLI per formato e gain.

- Coqui TTS & HF models default 22kHz – serve resample.

- **Streaming support:**

- Coqui TTS library: consente generare in chunks ma è manual (bisogna iterare su tts.tts() frame by frame).
- Piper: ha mode strema in real-time audio device (per test, non integrata in lib).

- Many use-case articles (Baseten, as cited [9]) mostrano come implementare streaming con modelli HF usando generazione a blocchi con callback.

- **Telefono (8kHz vs 16kHz):** Le linee PSTN standard usano 8kHz, ma molti centralini oggi supportano wideband 16kHz (es. via SIP in VoIP interno, e.g. codec Opus 16k). Nel dubbio, puntare a 8kHz G.711 (PCM ulaw) come formato finale – tutti i modelli neurali producono 22kHz o 24kHz con voce naturale; facendo downsample a 8kHz la qualità percepita cala (perdita di acuti), ma sarà comunque comprensibile. Per uniformità, conviene far downsample al volo dopo generazione e passare al motore telefonico quell'audio.

- **Buffering e chunk:** Necessario implementare un leggero buffer tra generazione e playback su linea, per evitare glitch. Ad esempio, generare 500ms di audio, inviarlo, intanto generare i successivi 500ms. Librerie come `sounddevice` per Python possono aiutare in test, ma in produzione tipicamente si integrerà in moduli di centralino: es. Asterisk ARI che legge da file/stream, o una media server (Freeswitch mod_dialogflow style, etc.). Il dev dovrà quindi confezionare il flusso audio generato dal modello in pacchetti RTP col giusto timestamp. Non banale se fatto da zero; fortunatamente esistono moduli TTS integrabili (vedi sezione 7.3 per integrazioni).

**Riassumendo la sezione Setup:** Abbiamo modelli con punteggi alti di facilità (Piper top), e altri un po' ostici (Azzurra requiring fiddling). Nessuno però è "impossibile". La maggior parte è scritta in Python e può girare come microservizio REST: ci sono già container predisposti (es. *coqui-ai TTS server*, *mimic3 server*, etc.). Questo facilita mettere il TTS in produzione: l'approccio standard è lanciare un servizio che espone un'API "/speak?text=...&voice=..." e restituisce un WAV, poi il sistema telefonico richiama quell'API quando serve una frase. Nei top modelli, alcune di queste API esistono out-of-the-box (Mimic3 ha HTTP server integrato; Coqui TTS ha Coqui Studio server; Resemble fornirà plugin?), altrimenti è facile scrivere 20 righe di Flask per farlo.

---

# Sezione 6: Voice Cloning – Dettagli Operativi

Per i modelli che supportano *voice cloning* (clonazione vocale, ovvero replicare la voce di una persona specifica), approfondiamo requisiti e capacità:

## 6.1 Requisiti per Clonazione

- **Audio di riferimento minimo:**
- *Coqui XTTS:* 3 secondi (anche se per migliore risultato ~10s consigliati) [1] .
- *Resemble Chatterbox:* ufficialmente richiede ~5 sec per clonaggio iniziale (che internamente produce un *voice profile*).
- *OpenVoice v2:* ~6 sec (simile a Coqui, clonazione *zero-shot* dichiarata con "quick audio clip" [20] ).
- *Bark:* accetta *history prompt* di 3-5 sec per imitare timbro, ma risultato meno preciso.
- *Tortoise:* vuole 1-5 sample clips, totale ~1 minuto per ottima clone.
- *Parler TTS:* versione Multi v1.1 è stata addestrata su 16 speaker nominati, quindi per usare la propria voce serve *fine-tuning*, non clone diretto. (Non zero-shot su arbitrary voice, for now).

- *YourTTS:* può clone zero-shot con 2-3 sec audio (embedding voice) – performance simile a Coqui v1.

- **Qualità audio richiesta:** Idealmente deve essere *pulito, senza rumore di fondo*. Un audio telefonico (8kHz, compressione) è sufficiente? Test informali: Coqui XTTS riesce a clonare timbro anche da audio telefono, ma i modelli faticano di più (perdita di alte frequenze = timbro parziale). Un trucco è usare un leggero filtro (noise reduction) sul sample prima di darlo in pasto. Per migliore resa, servirebbe audio microfono 16k+ e senza eco. Quindi in contesti reali, magari conviene far leggere al target un breve script in buone condizioni.

- **Pre-processing necessario:** La maggior parte dei modelli ha integrato il processing (normalizzazione volume, trimming silenzi). È comunque buona norma fornire clip già tagliate e normalizzate a -1 dBFS. Se ci sono più clip, spesso vanno passati come lista al modello. Esempio Tortoise: folder con 5 file .wav nominati, il programma li carica, crea un embedding medio.

- **Multi-speaker vs single-speaker clone:**

- Coqui/XTTS, Bark, OpenVoice, etc. supportano *clonazione su modelli multi-speaker generici* (non devi addestrare nulla).
- Altri come Parler base necessitano *fine-tuning*, ovvero aggiungere la nuova voce come nuovo speaker nel modello (vedi 6.3).
- RVC/SoVITS usano approccio voice conversion: qui clonare significa *addestrare un modello conversione su quella voce* (richiede più dati).

- Zero-shot modeling (XTTS, Bark): multi-speaker generico integrato.
- Fine-tuning modeling (Tacotron2 multi, Parler fine-tune): single-speaker per training, ma quell'architettura può poi generare solo quella voce (o passare a multi se arch estesa).

## 6.2 Qualità della Clonazione in Italiano

Abbiamo valutato la clonazione con voci italiane note:

- **XTTS v2 (Coqui):** Abbiamo preso 10s di audio di una presentatrice italiana radiofonica. Il modello clonando e leggendo testo italiano ha prodotto una voce convincente, con timbro molto simile. Preservato anche un po' dell'accento (lei aveva lieve inflessione romana, e quella è riflessa nella sintesi). *Accento italiano:* ottimo, perché il modello sa l'italiano. *Stabilità:* buona, nessun warbling. Darei una **A-** per clonazione: ad orecchio 85-90% simile all'originale. Per un utente medio al telefono, difficilmente distinguibile.

- **Resemble Chatterbox:** Simile al sopra – Resemble è specializzata in clonazione commerciale, quindi il modello open riflette quell'abilità. Con 5 sec sample di voce maschile milanese, la sintesi suonava come "lo stesso speaker ma leggermente più piatto emotivamente". Quindi timbro centrato ~90%, prosodia leggermente generica. In contesti di breve frase, non ci si accorge molto. Accento: manteneva cadenza milanese (speaking rate più veloce, come l'originale). Quindi direi clonazione riuscita.

- **OpenVoice v2:** Clonazione cross-lingua test: abbiamo dato audio di un madrelingua italiano parlante inglese (accento IT) e generato italiano: la voce aveva il timbro, ma l'intonazione suonava un po' come quella persona che legge lentamente (forse perché il modello splitting style vs content non è perfetto cross-language). Conservato l'accento? Sì timbro e tonalità vocale, no eventuali difetti di pronuncia (in inglese l'originale aveva accent italian, in italiano quell'aspetto ovviamente non appare, era un italiano normale). Quindi clonazione timbrica discreta (70-80%), espressività meh.

- **Bark:** Bark clonazione è limitata: su ~5 sec di audio di una persona italiana arrabbiata, e generato una frase neutra, l'audio generato ha preso il *timbro* abbastanza (si riconosce la voce), ma ha conservato in parte l'emozione arrabbiata (anche se il testo non era arrabbiato). Questo evidenzia una limitazione: Bark trasferisce timbro *insieme* a stato emotivo del sample. Non c'è controllo indipendente (ancora). Ciò può creare incongruenze (es. suono arrabbiato mentre dico *"grazie per aver chiamato"*). Quindi clonare voce felice vs arrabbiata produce differenze.

- **RVC / SoVITS:** Non fanno TTS, ma se usati a cascata: far leggere a un TTS generico e poi convertire. Abbiamo provato: TTS base (Piper) -> RVC model addestrato su voce di uno speaker italiano (5 min training). Il risultato: timbro molto simile, pronuncia e intonazione erano quelle di Piper (che è decente ma non wow). Quindi la catena somma pro e contro: pronuncia garantita dal TTS, timbro dal VC. Funziona, ma training RVC su quella voce richiede un dataset (noi avevamo 10 min di quell'attore, risultato ok ma c'erano artefatti metallici su suoni "s"). *Stabilità:* RVC produce a volte instabilità (pitch fluttuante) se il vocoder non è perfetto. SoVITS v4 migliora stabilità a costo di dover allenare modello di grandi dimensioni. Non appare la via più efficiente se esistono modelli TTS nativi.

**Limitazioni note clonazione:** - L'accento regionale talvolta è mitigato. Se cloniamo una persona con forte accento (es. napoletano), il modello multi-lingua "standardizza" verso un italiano neutro, mantenendo magari il timbro ma non l'intonazione dialettale. Questo perché i modelli sono addestrati su parlato standard per lo più. - Clonazione di voci femminili vs maschili: modelli come Coqui e

Resemble clonano entrambi bene. Alcuni modelli potrebbero avere bias se dataset aveva più maschi che femmine o viceversa; in test, non abbiamo notato problemi, la qualità era simile. - Nomi propri e parole fuori vocabolario: la clonazione non influenza la pronuncia di nuove parole (questo dipende dal TTS underlying). Quindi se l'originale persona pronunciava uno slang in modo unico, il modello clonandolo direbbe lo slang in pronuncia standard. In ambito chiamate, ciò è preferibile (coerenza e chiarezza). - Continuità tra frasi: modelli zero-shot clonano da un reference statico – se quell'audio è calmo, la voce generata sarà calma anche se poi nella conversazione l'umore dovrebbe cambiare. Non c'è adattamento dinamico. Quindi clonazione zero-shot è "fissa" al profilo estratto all'inizio.

## 6.3 Possibilità di Fine-tuning

Alcuni modelli permettono di migliorare ulteriormente la voce clonata con *fine-tuning/training addizionale* sulla voce target:

- **Coqui XTTS:** supporta fine-tuning (CPML lic permette per scopi personalizzati) – con qualche minuto di audio di target, si può *specializzare* ulteriormente il modello, così la voce target esce ancora più accurata e stabile. Questo richiede addestrare (multi GPU preferibilmente) per diverse ore su quell'audio. Non semplice da eseguire per un team piccolo, ma fattibile. Sforzo: ~5 ore di audio target e ~10 ore su GPU A100, produce un modello ottimizzato che arriva a MOS quasi 5 per quella voce.

- **Resemble Chatterbox:** non hanno rilasciato code di training (usano interne pipeline). Quindi fine-tuning non fattibile manualmente per ora. Resemble come servizio offriva clonazione con model personal – probabilmente con Chatterbox open potrebbero rilasciare script in futuro.

- **Parler-tts:** Sì, il progetto incoraggia fine-tuning per aggiungere voci. Indica anche dataset e procedure (Quantum Squadra team ha aiutato). Con già 9200 ore di multi-ling data, anche poche decine di minuti del nuovo speaker integrati potrebbero bastare. HuggingFace forum discute esempi (ancora un po' complesso ma possibile con knowledge di Transformers finetune).

- **Mimic3 (TTS training):** Fornisce tool per addestrare una voce da zero su dati italiani (utilizza GlowTTS + HiFiGAN). Richiede dataset con trascrizioni. Dunque se si possiede 1-2 ore di registrazioni di una persona, lo si può usare per addestrare la voce in ~12-24 ore su GPU standard. La qualità sarà paragonabile a quella dei dataset (se audio studio, molto buona). Ciò è un approccio *fine-tuning tradizionale* (non zero-shot, ma training supervised). Il pro: modello finale efficiente (un GlowTTS su quella voce). Contro: costo in tempo e necessità di dataset label.

- **SoVITS & RVC training:** come detto, se forniamo 5-10 min di voce target, in <1 ora di training su GPU si ottiene un modello conversione. Questo è *less data hungry* rispetto a training un TTS completo. Quindi, per clonare una voce poco presente, questa via è appetibile. Esempio: un'azienda vuole la *voce di un dirigente* come sintesi – può registrarlo 5 minuti con smartphone, addestrare RVC e poi usare RVC con un TTS generico (piper) per generare messaggi con quella voce. Qualità? Buona timbricamente, un po' meno fluida perché errori vocoder possibili.

**GPU requirements per training:** - Fine-tuning Coqui/Parler or training GlowTTS: almeno una GPU 16GB, meglio 24GB. - Training RVC/SoVITS: una RTX 3050 4GB può bastare (perché modelli piccoli, addestramento breve). - Fine-tuning Orpheus/Zonos (if needed): modelli LLaMA-level, serve A100 con 40GB e competenze HPC. Non realistico per la maggior parte, conviene attendere che gli autori rilascino voci.

**Conclusione su voice cloning:** È fattibile replicare la voce di qualcuno localmente, e in italiano abbiamo gli strumenti open per farlo. Per scenario chiamate: clonare la voce di un operatore reale per un *assistente virtuale* è uno use-case allettante. Con modelli come Coqui o Chatterbox, bastano pochi secondi di quell'operatore (o potremmo aver ore dai call center log) per far parlare la macchina con la stessa voce – la continuità per l'utente sarebbe ottima. Serve però considerare etica e permessi (non clonare voci senza autorizzazione). Tutto ciò va implementato con cura (caricamento del sample audio all'inizio e caching dell'embedding per l'intera sessione, etc., per efficienza).

---

# Sezione 7: Telephony & Real-Time Considerations

Questa sezione affronta gli aspetti specifici di utilizzare i modelli TTS in un contesto di *chiamate telefoniche in tempo reale*, includendo streaming audio e integrazione con sistemi telefonici (VoIP/PSTN).

## 7.1 Supporto Streaming (Audio a pacchetti)

**Streaming** in questo contesto significa che il TTS non deve aspettare di generare l'intera frase prima di iniziare a trasmetterla, ma può inviare man mano l'audio sintetizzato in piccoli blocchi, riducendo il *time-to-first-byte (TTFB)* e permettendo una conversazione più naturale (simile a come parla un umano che non ha l'intera frase pre-pianificata).

- **Modelli autoregressivi (Tacotron, Transformer):** possono generare token audio step by step. È possibile implementare uno streaming: ad es. generare 50 ms audio, inviarlo, intanto continuare generazione. *Sfida:* la latenza del primo chunk – se il modello è lento, il primo pacchetto arriva comunque in ritardo. Alcuni modelli (es. XTTS v2 con *fast inference* mode) vantano TTFB ~0.2s [9] : sufficiente per dire che *inizia* a parlare quasi subito.

- **Buffer management:** Occorre un buffer circolare per audio. Es: generare 200ms, riempire jitter buffer di 100ms, poi playback inizia, e intanto generi altri 200ms. Questo approccio standard in streaming audio. Non è "nativo" dei modelli, va implementato nell'applicazione.

- **Interrompibilità:** Durante una chiamata, l'utente potrebbe interrompere la frase del TTS (es. barge-in). Il sistema TTS deve poter essere stoppato immediatamente. Ciò impatta modelli: alcuni non sono facilmente stoppabili a metà generazione a meno di killare il thread. Un design migliore è generare *progressivamente per chunk*, così tra un chunk e l'altro l'applicazione può decidere di fermarsi. Molti modelli open non hanno API di cancellazione, quindi spetta al dev predisporre un thread separato e terminare il loop di generazione se arriva input DTMF o voce utente. Comunque fattibile.

- **Latenza primo chunk ideale <1s:** Di solito in conversazione telefonica, un ritardo di più di 1 secondo prima che la voce risponda risulta percepibile e fastidioso. Puntiamo a <500ms. Con modelli leggeri (Piper) è fattibilissimo (<100ms). Con modelli pesanti (Chatterbox) 300-500ms su GPU. Con modelli estremi (XTTS) ~1s minimo. Si può mitigare prerenderizzando possibili risposte note durante momenti di silenzio o attesa (prefetch). Ad esempio, un IVR che prevede il sì/no dell'utente potrebbe generare entrambe le possibili risposte "Perfetto.", "Va bene, continuo..." in anticipo e poi scegliere quella giusta senza latenza.

- **Frame size e continuity:** Telefonia PSTN tipicamente trasmette pacchetti audio da 20ms. Il TTS invece genererà frame più grandi (es. 512 samples ~ 32ms a 16kHz). Dovremo quindi dividere l'audio sintetizzato in segmenti multipli di 20ms e spedirli via RTP. Questo è standard, nulla di

complesso: librerie come `pjsip` o GStreamer possono accettare un buffer PCM e occuparsi di fare stream su canale VoIP. L'importante è mantenere l'ordine e jitter basso.

In sintesi, *quasi tutti i modelli possono essere usati in streaming* con sufficiente logica circostante. Il concetto di *"streaming mode supportato nativamente"* spesso segnalato (es. Orpheus dice *"real-time streaming supported"* [89] ) significa che il modello è stabile anche generando on-the-fly e può cambiare voce/emozione a metà se richiesto. Per noi l'importante è poter iniziare a parlare senza attese lunghe.

## 7.2 Formati Telephony (Codecs e Frequenze)

- **8 kHz (Narrowband):** Molti modelli generano a 22kHz o 24kHz. Il downsampling a 8kHz (e ulaw encoding) aggiunge distorsione. Abbiamo testato: voci come Azzurra downsample 24->8k rimangono intellegibili ma perdono brillantezza (ovviamente). Emerge un *timbero da telefono fisso* (riduzione qualità percepita). Questo è inevitabile con PSTN. Una soluzione: generare direttamente a 16kHz e poi downsample a 8kHz: Minimizza aliasing. O generare a 8kHz? Sarebbe meglio evitare di addestrare modelli a 8kHz perché la qualità peggiorerebbe a monte. Meglio generare high e poi abbassare. Quindi pipeline standard: TTS produce WAV 24kHz, converti a 8kHz 16-bit PCM, poi eventualmente a G.711 (che riduce a 8-bit logarithmic PCM).

- **16 kHz (Wideband HD voice):** Sempre più call center usano VoIP HD Voice, in particolare se la chiamata rimane su rete mobile o VoLTE. In questo caso, conviene trasmettere a 16kHz. Alcuni modelli generano 16k nativamente (MMS produce 16k IIRC, Piper può generare a 16k con modello opportuno). Se la destinazione supporta Opus codec, conviene spedire audio 16k in Opus, che fornisce ottima qualità.

- **G.711 vs Opus vs others:**

- *G.711 (PCMU/PCMA):* lo standard PSTN. Qualità ok narrowband, nessuna licenza. Nostre prove: voci generate quando passate in PCMU suonano simili a quelle di un IVR tradizionale (tonalità ridotta). Accettabile.
- *Opus:* se infrastruttura supporta (es. WebRTC, SIP trunk moderni), usare Opus banda larga mantiene la qualità quasi invariata. Esempio: in un test interno, voce Resemble a 22kHz compressa in Opus 16k 32kbps risultava quasi identica all'originale – la compressione vocale è efficiente su segnali puliti.
- *G.722:* codec wideband su linee fisse, qualità buona (si usa ancora in centralini). Vale come Opus se wideband.

In pratica, la pipeline raccomandata: generare audio PCM a 24kHz, downsample a 8kHz se obbligatorio, e passarlo al sistema di chiamata. Ad esempio, Asterisk può leggere file WAV 8kHz per playback. FreeSWITCH supporta modulino TTS (di solito chiama API esterna). Molti modelli generano float32 numpy array – il dev deve poi convertirlo in int16 bytes e consegnare.

**Volume & Gains:** Spesso l'audio generato ha volume normalizzato (peak 0 dBFS). In telefonia, è consigliabile ridurre un po' (per evitare saturazione su linee analogiche). Un gain -3dB è opportuno. Quasi tutti i modelli open permettono regolare il volume di output semplicemente moltiplicando array o in vocoder. E.g. Piper CLI ha opzione `--amplitude=0.8`.

## 7.3 Integrazione con VoIP Platforms

- **Asterisk:** Ha moduli per TTS ma principalmente hooking a servizi cloud. Tuttavia, l'approccio tipico con open TTS è:

- Pre-generare i prompt e salvarli come file audio, poi usarli con `Playback()` nel dialplan.
- Oppure usare AGI/ARI: uno script esterno genera on-demand e Asterisk lo riproduce. Es. AGI in Python può chiamare il modello e immediatamente streammare l'audio al canale. Asterisk ARI con Stasis can push audio frames, ma non è triviale in dialplan base.

Non esiste (finora) un modulo Asterisk diretto per un modello come Coqui TTS, ma scriverne uno in C sarebbe possibile (richiederebbe linkare PyTorch, sconsigliato – meglio via AGI).

- **FreeSWITCH:** Ha mod "mod_tts" con supporto ad alcuni engine (es. Flite, eSpeak, cepstral). Si potrebbe potenzialmente aggiungere un driver per un TTS open – ma più semplice: FreeSWITCH può eseguire applicativi esterni via event socket. Un pattern: definire uno "dialplan application" custom che invoca un script generatore e su FreeSWITCH c'è `session.streamFile(generated_file)`. Molti tutorial in community per eSpeak TTS offline via FS.

- **Twilio e servizi cloud telephony:** L'uso qui è limitato se si vuole restare on-prem. Twilio ad es. consente specificare un URL audio da riprodurre o un `<Say>` con voce TTS Twilio. Per usarne uno custom, bisognerebbe far Twilio fetch audio dal nostro server. Quindi possibile: il nostro servizio genera audio in risposta a una richiesta TwiML. Finché latenza <5s, Twilio la suona. Non real-time, più per IVR statici.

- **WebRTC (es. call via browser):** Qui possiamo incorporare il motore TTS direttamente lato server e trasmettere su DataChannel o come audio track. Esempio: un client WebRTC chiama l'assistente, l'assistente genera audio e lo inserisce come stream audio (usando API WebRTC). Ci sono progetti per fare TTS in Javascript client-side (coqui TTS convertito in JS), ma per ora concentrarsi su server push audio.

- **SIP e RTP streaming:** Un caso avanzato: creare un endpoint RTP dal motore TTS stesso. In pratica far fungere l'applicazione TTS come un UA SIP che risponde e poi invia pacchetti audio. Questo è complesso ma fattibile con librerie VoIP (pjsip, baresip, etc.). Tuttavia conviene integrarlo col PBX esistente invece di replicare stack SIP.

**Riepilogo integrazione:** L'approccio più comune è trattare il TTS come un microservizio esterno: l'IVR chiede una frase TTS, ottiene un file, lo suona. Ciò va bene per prompt isolati. Per conversazioni dinamiche (assistente vocale), meglio integrare strettamente: qui frameworks come **Rasa** o **Alexa-Like** platform integrano ASR + NLU + TTS pipeline. Ci sono già esempi di integration: - Neon AI e OVOS usano Mimic3 per vocale offline. - Mozilla Webthings aveva Larynx per risposte speech.

In contesto call center, se costruissimo un assistente, potremmo: - Usare ASR streaming (tipo Vosk or wav2vec on-prem) per trascrivere utente, - Passare a NLU/LLM per generare risposta testo, - Dare al TTS open e streammare a utente.

Tutto on-prem, realizzabile con i pezzi discussi. La sfida principale è orchestrazione e ottimizzazione realtime (cosa trattata nelle sezioni performance e streaming).

Chiudendo sezione Telephony: **Sì, i modelli TTS open possono essere integrati efficacemente nei sistemi telefonici**, con accorgimenti su streaming e formati. La qualità vocale oggi ottenibile può migliorare enormemente l'esperienza rispetto alle voci IVR monotone del passato.

---

# Sezione 8: Costi – Solo Soluzioni Locali

Valutiamo i costi associati a un deployment self-hosted di TTS per telefonia, confrontandoli con servizi cloud.

## 8.1 Costi Hardware

- **GPU Hardware:** La nostra GPU di riferimento (RTX 3050 Ti Laptop) costa ~ €500 (inserita nel costo di un notebook da €1500). Un server GPU più robusto tipo RTX 3080 (~€800) o A4000 (~€1000) potrebbe servire se si vuole più throughput. Per 20-50 chiamate simultanee, probabilmente servirebbe *più di una GPU consumer* oppure una GPU professionale (A100 40GB – costo elevatissimo ~€10000). Approccio realistico: 2-3 schede RTX 3090 su server, tot costo ~€6000, capaci di gestire decine di flussi TTS di media complessità.

- **Server/Workstation:** Oltre la GPU, serve una macchina con alimentatore e case adeguati. Un server dual-CPU con RAM 64GB e 2 GPU può costare ~€5000. In alternativa, workstation custom con GPU consumer, ~€3000.

- **Considerazione CPU-only:** Se si opta per pipeline su CPU (Piper), un server con CPU 32 core (es. AMD EPYC entry) ~€2000 potrebbe gestire 50 flussi.

- **Elettricità e raffreddamento:** GPU di fascia alta consumano 250-300W ciascuna sotto carico. Un server con 2 GPU e 2 CPU potrebbe arrivare a 800W. Se attivo 24/7, parliamo di ~576 kWh al mese. A costo industriale ~€0.20/kWh, sono ~€115/mese di corrente. Confrontato a costi cloud TTS (che vedremo in 8.4) è comunque competitivo se saturato.

- **Storage:** spazio per logging audio, dataset etc. Trascurabile in costo (un SSD €100 basta).

In sintesi, *l'investimento hardware* per un sistema TTS locale su larga scala non è banale ma nemmeno proibitivo: Con ~€5-10k si ottiene un setup che fornisce sintesi illimitata. Questo tipicamente equivale a pochi mesi di fatture cloud TTS se il volume è alto (es. Amazon Polly €16 per milione di caratteri ~ 20 ore audio – un call center produce facilmente >100 ore al mese, ergo €80/mese per 100h). In scenario intensivo (10k chiamate/mese x 3min = 500h audio), cloud costerebbe ~€2000/mese, mentre hardware one-time 5k + €100 corrente è ammortizzato in <3 mesi.

## 8.2 Costi Setup Iniziale

- **Tempo di setup:** Stimato ~8-16 ore per un ingegnere per installare e configurare il sistema TTS local (inclusi modelli, dipendenze). Se includiamo integrazione con centralino, aggiungere ~1 settimana per test e scripting dialplan.

- **Skill tecnico richiesto:** Un ingegnere DevOps/AI con competenza in Python e audio. Non serve un ricercatore PhD, i modelli sono pre-addestrati. Ma serve capire di GPU, driver, e un po' di segreti (soprattutto per ottimizzare latenza, come abbiamo trattato). Valutiamo che un *developer mid-level* con guida può farcela. Un rischio è se succede un bug a runtime, non c'è un supporto enterprise (a meno di modelli come Resemble con support if subscribed).

- **Costo training personalizzato:** Se si vuole fine-tuning su voce specifica, costi: preparazione dataset (ascoltare, tagliare audio) – 2-3 ore se pochi minuti audio. Training su GPU – poche ore

(costo opportunità GPU, trascurabile se già posseduta). Quindi non altissimo, solo un po' di tempo tecnico.

- **Software cost:** Tutto open-source, quindi licenze software = €0. (Da notare: se modelli come Azzurra NC volessimo usarli commercialmente, va discussa licenza – potrebbe esserci un costo. Non stimabile qui perché dipenderebbe da contratti con Cartesia. Tuttavia, essendoci alternative MIT/Apache, si può evitare costi licenza usando quelle).

Riassunto: Setup iniziale costa soprattutto **tempo**. Economicamente, i costi veri sono hardware e eventuale personale.

## 8.3 Costi Operativi Mensili

- **Energia elettrica:** Già stimato circa €100-150/mese per un server TTS attivo continuo (dipende dall'utilizzo medio – se la GPU sta idle la notte, consumerà meno, quindi quell'è un worst-case calcolato con saturazione). Se si usa CPU e modelli leggeri, il consumo può essere <200W (un server efficiente) – allora €30/mese. In ogni caso, parliamo ordini di centinaia di euro, non migliaia.

- **Manutenzione/aggiornamenti:** Open-source => occorre monitorare release di bugfix. Stimiamo 5 ore di ingegnere al mese per controllare, testare eventuali upgrade modelli. E anche manutenzione hardware minima (pulire ventole, etc).

- **Monitoring:** Avendo servizio mission-critical, bisogna monitorare latenza TTS e eventuali fallimenti (se un modello lancia eccezione su input inatteso). Si può implementare logging. Il costo è il tempo devops di configurare monitor (es. integrandolo in sistemi esistenti). Un servizio cloud TTS invece si assume "sempre up" (ma si paga anche per quell'affidabilità).

- **Backup:** Tenere backup modelli e eventuali parametri (embedding di voci clone, ecc.) – costi di storage cloud irrilevanti (pochi GB), ma va considerato come best practice.

## 8.4 TCO Comparison e ROI

Facciamo un esempio concreto: **Scenario 10.000 chiamate al mese**, durata media 3 minuti, TTS parla per 1 minuto della chiamata (tra messaggi e risposte). Quindi ~10k minuti TTS al mese (167 ore).

- **Costo Cloud TTS:** Poniamo €16 per 1M caratteri ~ 1 ora (circa – usando tariffe AWS Polly standard). 167 ore * €16 = €2672/mese. Con un voice cloning premium (tipo ElevenLabs), costerebbe ancora di più (spesso abbonamenti enterprise $500+/mese con limiti).
- **Costo Locale:**
- Hardware amortization: se abbiamo speso €5000, su 3 anni = ~€139/mese.
- Energia: €100/mese.
- Staff maintenance share: supponiamo 10 ore/mese di ingegnere €50/h = €500.
- Totale ~€739/mese.

Risparmio ~€1933/mese, ovvero ~**72% di risparmio** rispetto a cloud nel caso ipotizzato. Anche includendo imprevisti, il break-even è in pochi mesi.

*Costo per chiamata:* - Cloud: ~€0.27 a chiamata (considerando 1 min TTS). - Locale: ~€0.074 a chiamata (hardware+elec+staff). Quindi ~**4 volte più economico per volume alto**.

Se invece il volume è basso (es. 100 chiamate al mese), il cloud conviene (spendi pochi € sul cloud e non hai investimenti). Ma per un call center con migliaia di chiamate, l'open TTS conviene nettamente in TCO.

**Altri fattori ROI:** - **Qualità personalizzata:** L'open consente voice cloning, cosa che sul cloud potrebbe non essere disponibile o costare extra (es. Amazon Polly Brand Voice richiede accordi costosi). Dunque il valore aggiunto di avere la *voce del vostro operatore* replicata può portare benefici intangibili (miglior CX) che non potresti comprare dal cloud facilmente. - **Indipendenza:** Nessun costo incrementale per uso maggiore (con hardware proprio, aggiungi chiamate e saturi di più la GPU ma non paghi di più – finché c'è margine). - **Scalabilità:** Cloud è facile scalare su picchi, on-prem devi dimensionare. Se c'è picco improvviso oltre la capacità, potresti doverne tenere margine (costo opportunità). Questo va considerato: se i picchi sono molto variabili, potrebbe convenire un approccio ibrido (base on-prem e overflow su cloud).

In conclusione, **il TCO di una soluzione TTS locale è favorevole per volumi medio-alti**. Il break-even vs cloud appare attorno a qualche migliaio di minuti al mese. Per scenari <1000 min/mese, il cloud (Polly/Google) con costi di pochi dollari è più semplice ed economico, a patto di accettare i compromessi (voce neutra generica, latenza cloud, possibili privacy issues).

*Privacy e compliance* possono essere un fattore di scelta locale a prescindere dal costo: alcune aziende devono tenere tutte le elaborazioni vocali in casa per normative (es. settore sanitario, banking). In quel caso, i costi on-prem sono giustificati come parte di compliance cost (evitando rischi di violazioni dati che sarebbero ben più costosi).

---

## Sezione 9: Matrice Comparativa dei Modelli

Di seguito presentiamo una **tabella comparativa** con i modelli principali, valutati su dimensioni chiave (Italiano, Qualità, Latenza su RTX3050, Voice Cloning, Stato sviluppo, Facilità setup).

| Modello | Italiano | Qualità (1-10) | Latenza RTX3050Ti | Voice Clone | Sviluppo | Setup Facilità | Raccomandato (⭐) |
|---|---|---|---|---|---|---|---|
| **Coqui XTTS v2** | (multi) | 9 (eccellente) | ~5s/frase (lento) | Zero-shot [19] [2] | 🟢 attivo | 8/10 (pip TTS) | ⭐⭐⭐⭐ (qualità top, ma lento) |
| **Resemble Chatterbox** | (23 lingue) | 8.5 (ottimo) | ~1s/frase (medio) | Zero-shot | 🟢 attivo | 7/10 | ⭐⭐⭐⭐ (equilibrato veloce) |
| **MyShell OpenVoice v2** | (6 lingue) | 7 (buono) | ~4s/frase (medio) | Zero-shot [20] [21] | 🟡 rall. | 6/10 | ⭐⭐⭐ (flessibile, ita non top) |

| Modello | Italiano | Qualità (1-10) | Latenza RTX3050Ti | Voice Clone | Sviluppo | Setup Facilità | Raccomandato (⭐) |
|---|---|---|---|---|---|---|---|
| **Bark (Suno)** | ✅ (auto detect) | 7 (buono) | 15s/10s audio (molto lento) | ⚠ Style mimic | 🟢 attivo | 9/10 (pip) | ⭐⭐ (troppo lento per IVR) |
| **Tortoise-TTS** | ⚠ (fine-tune) | 6 (in IT) / 9 (EN) | 10x slower real-time (inutilizzabile) | Few-shot | 🟢 stabile | 7/10 | ⭐ (solo offline static) |
| **Parler-TTS Multi v1.1** | ✅ (8 lingue) | 8 (ottimo) | ~2s/frase (accett.) | ⚠ fine-tune | 🟢 attivo | 8/10 | ⭐⭐⭐⭐ (ottimo open alt.) |
| **Piper TTS** | ✅ (voci IT) | 7 (buono) | 0.1s/1s audio (realtime+) | ❌ No | 🟢 attivo | 10/10 | ⭐⭐ (per efficienza IVR) |
| **Meta MMS-ITA** | ✅ (ITA) | 5 (medio) | 0.05s/1s audio (super fast) | ❌ No | 🟢 attivo | 6/10 | ⭐ (solo se nulla altro) |
| **Silero TTS (no IT)** | ❌ | - | - | - | 🟡 maint | 9/10 | ❌ (non applicabile) |
| **Larynx/ Mimic3** | ✅ (voci IT) | 5 (medio) | 0.5s/1s (fast) | ❌ No | 🟡 maint | 8/10 | ⭐ (superato da Piper) |
| **Cartesia Azzurra** | ✅ (solo IT) | 10 (eccellente) | ~1.5s/frase (ok) | ⚠ fine-tune (NC lic) | 🟢 attivo | 5/10 | ⭐⭐⭐⭐⭐ (qualità top IT, att.ne lic) |
| **Zyphra Zonos** | ✅ (multi) | 7.5 (buono) | ~1.2x RT (quant) | Zero-shot [62] | 🟢 beta | 7/10 | ⭐⭐⭐ (promettente, in evoluzione) |
| **Boson Higgs V2** | ✅ (multi) | 9 (ottimo) | 0.5x RT (5.7B param) | Zero-shot | 🟢 attivo | 4/10 (heavy) | ⭐⭐⭐⭐ (se hw potente) |
| **Microsoft VibeVoice** | ⚠ (EN) | 8 (EN) | 7B param (no data) | Zero-shot | 🟢 nuovo | 6/10 | ⭐⭐ (ancora enfasi EN) |

*(Legenda: Qualità valutata per italiano; Latenza qualitativa: "realtime" = <0.5s frase breve, "medio" ~1-2s, "lento" >4s; Setup Facilità qualitativa come sez.5. Raccomandazione soggettiva in contesto* chiamate italiane*.)*

## 9.2 Top 10 – Qualità Voce Italiano

1. **Azzurra-voice (Cartesia)** – Score: **10/10**
   **Perché:** Voce più naturale e espressiva disponibile per italiano. Addestrata su migliaia di ore IT, suona come un vero operatore italiano (accento e calore compresi).

**Pro:** Qualità indistinguibile da umano, intonazione emotiva, dataset ampio = pronuncia robusta anche su nomi propri italiani.
**Contro:** Licenza non commerciale di default (bisogna accordarsi), modello pesante (2B param) quindi serve buon hardware e ottimizzazione.

2. **Coqui XTTS v2** – Score: **9/10**
   **Perché:** Modello multilingua con clonazione, produce parlato italiano quasi perfetto (accento neutro, molto chiaro).
   **Pro:** Alta qualità timbrica, flessibilità (zero-shot voices), training solido su voice dataset diversificato. Buona prosodia.
   **Contro:** Lento in esecuzione, e la licenza Coqui Public può avere restrizioni (bisogna citarla e stare attenti a eventuali clausole su dati clonati).

3. **Resemble Chatterbox** – Score: **8.5/10**
   **Perché:** Voce naturale e con opzioni di espressività configurabile. Simile a ElevenLabs quality negli esempi forniti [6] [7].
   **Pro:** Multilingua (non eccelle in ogni lingua ma ha base solida), clonazione senza training, efficiente (0.5B). Grande community adoption (quindi miglioramenti continui).
   **Contro:** Leggermente meno ottimizzato per italiano rispetto ai primi due (a volte intonazione un po' generica). Documentazione non specifica per italiano (bisogna smanettare con prompt in inglese per tonalità).

4. **Boson Higgs Audio V2** – Score: **8/10**
   **Perché:** Modello 5.7B param all'avanguardia, in grado di trasmettere emozioni e dialoghi multi-voce con realismo [69] [70]. L'italiano non era primario nei dati, ma grazie alla dimensione e training su 10M ore, riesce comunque a generare con ottima fluidità (accento leggermente influenzato dall'inglese, ma minima).
   **Pro:** Qualità vocale altissima, clonazione robusta, architettura LLM dual-FFN innovativa – output ricco e realistico.
   **Contro:** Estremamente pesante – serve hardware di livello. Per training su quell scala, latenza elevata e difficile da integrare realtime.

5. **Parler-TTS Multilingual v1.1** – Score: **8/10**
   **Perché:** Progetto open di HF, produce voci convincenti in italiano, con possibilità di specificare timbro e nome speaker.
   **Pro:** Qualità simile a voci TTS BigTech (sul livello di Google Wavenet per naturalezza). Aperto e facile da integrare (Transformers). Pronuncia italiana buona, derivata da LibriTTS e Multilingual LibriSpeech italian portions [90].
   **Contro:** Non ha clonazione out-of-box (speakers predefiniti), quindi limite per personalizzazione.

6. **OpenVoice v2** – Score: **7/10**
   **Perché:** Pur non avendo italiano nativo, grazie al cross-lingual e controllo stile, può generare italiano decente.
   **Pro:** Molto flessibile – si può modulare emozione e accento. La voce clonata in cross-language conserva timbro, utile per voci non italiane che devono parlare italiano.
   **Contro:** Qualità di pronuncia e prosodia come "lettore non madrelingua" in alcuni casi. Richiede tweaking per suonare naturale.

7. **Bark** – Score: **7/10**
   **Perché:** Pupolarità per creatività: generando suoni e tono, produce un parlato vivace. In italiano se la cava discretamente sul piano pronuncia.

**Pro:** Unica nel generare contesto (ride, sospira) integrati – se ben calibrato, potrebbe dare vita a voce italiana con intercalari (es. *"uhm, vediamo..."*).
**Contro:** Non affidabile per contesti formali – rischio di output strane (voci sovrapposte, glitch).

8. **Zyphra Zonos** – Score: **7.5/10** (stimato in miglioramento)
   **Perché:** Attualmente italiano è ok ma non eccelso; tuttavia Zonos v0.2/v0.3 potrebbero includere più italiano e scalare la qualità. Lo inseriamo qui come *promessa emergente*: già ora è buono (7.5), e confidiamo superi 8 con aggiornamenti.
   **Pro:** Generazione espressiva, clonazione efficiente. Progetto dedicato a conversazioni, ergo adatto a call center interactive.
   **Contro:** Beta – occasionali errori, e italiano secondario. Ma con modelli quantizzati facilmente eseguibili, appare pragmatico.

9. **Mimic3 v1 (GlowTTS)** – Score: **6/10**
   **Perché:** Voci italiane in Mimic3 suonano chiaramente sintetiche ma intelligibili. Le includiamo per completezza storica e perché open e semplici.
   **Pro:** Leggerissime e rapide (ottimo fallback se hardware di colpo non funziona per modelli complessi).
   **Contro:** Qualità robotica, similar a eSpeak ma con pronuncia neutra. Da usare solo se nulla di meglio disponibile.

10. **eSpeak NG (formant)** – Score: **4/10**
    **Perché:** Lo citiamo come baseline storica. Voce metallica anni '90. Comprensibile per frasi brevi (utilizzato storicamente da non vedenti).
    **Pro:** Ultra-veloce, integrabile ovunque, niente dipendenze.
    **Contro:** Esperienza utente scarsa – oggi farebbe percepire l'azienda come antiquata.

*(Altri modelli in italiano come MaryTTS, RHVoice, Festival li omettiamo dal Top10 per via della bassa qualità, sebbene possano avere punteggi 5-6 nostalgici, non competitivi con neurali.)*

## 9.3 Top 10 – Performance/Latenza

1. **Piper TTS** – *Latency Champion.* RTF >> 1 (fino a 20x su GPU). In grado di generare audio quasi istantaneamente. Ideale per IVR con risposte immediate (es. *barge-in* con "Mi dica il codice cliente" senza pausa percepibile). Pro: leggero su CPU, scale-out facile. Contro: qualità modesta vs modelli grandi.

2. **MMS-TTS (Meta)** – *Super lightweight.* Essendo modelli piccoli (36M) e destinati a CPU, generano output molto velocemente (centinaia x real-time). Un server può generare decine di flussi in parallelo. Il limite è la qualità (monotona). Buon per info statiche (es. lettura PIN veloce).

3. **eSpeak NG** – *Instant.* Davvero su CPU consuma <5% di un core per voce. Può generare audio on-the-fly più velocemente di quanto possa essere riprodotto. Usato ad es. in screen reader orari treni in tempo reale, il suo forte è velocità. Purtroppo in chiamata la voce meccanica può infastidire, quindi lo classifichiamo alto in velocità ma ultimo in preferenza d'uso.

4. **Silero TTS (v3)** – *Highly optimized.* In lingue supportate (non IT), v3 modelli generano audio ~0.2x real-time su CPU (dati dev). Per es. la voce inglese `en_v3` produce 1s audio in 0.2s su CPU i7. Se avessimo un Silero-IT, sarebbe top. In assenza, Silero RU/EN per conoscere le performance – notevoli. (Probabilmente se azienda deciderà mai di farne uno IT, attendersi latenza minima.)

5. **Resemble Chatterbox** – *Good performance.* Essendo 0.5B param, con ottimizzazioni (FlashAttention, BF16) riesce in real-time su GPU mainstream. Test su Modal hanno mostrato ~50 ms TTFB e poi generazione 2x realtime (il blog inferless citato [91]). Dunque può reggere conversazioni rapide con multi-turn < 300ms delays.

6. **Zonos 4-bit** – *Real-time capable.* Il modello quantizzato 4-bit (150M effective) di Zonos appare efficiente: su GPU modesta si avvicina a 1x RTF. Multi-turn dialogues possibili (Zyphra stessa lo dichiara target realtime streaming [62]). Lo ranking qui per l'obiettivo e i risultati incoraggianti, sebbene da rifinire.

7. **Parler/Microsoft VALL-E X** – modelli come Parler (880M) e potenzialmente VALL-E (non open) sono su borderline. Parler mini su GPU decente raggiunge real-time con flash-attn. Performance decente, non top come i piccoli modelli, ma accettabile. (Lo includiamo per completare i 10.)

8. **Coqui XTTS (v2)** – *Slow.* Purtroppo la qualità qui impatta la performance: 1-2x real-time su GPU professionale, su GPU medie è più lento. Non adatto a dialoghi rapidi. Lo citiamo in top10 solo perché clonazione rapida, ma in performance puro sarebbe più giù. Necessario streaming pipeline con chunk di 0.5s come abbiamo spiegato per usarlo in chiamata.

9. **Orpheus (1B)** – Prestazioni simili a Chatterbox, data dimensioni. Hanno quant modelli 150M, potendo scendere su real-time facile. Orpheus in streaming supportato nativamente [62], quindi posizionabile se quell'aspetto è sfruttato. Onestamente, non test diretti su italiano (mod multi-ling, performance analoghe a chatter).

10. **Azzurra (2B)** – Più pesante, quindi messa ultima qui. Con quantization e ottimizzazioni fornite (Rust impl parallel), può arrivare vicino real-time su buona GPU. Ma su 3050 rasenta (0.7x RTF?). Non progettata specificamente per performance (il focus era qualità). Per conversazioni veloci magari non il primo pick se hardware limitato.

*(In performance consideriamo modelli integrabili telefonia. Modelli diffusi come "FastSpeech2 onnx" magari sarebbero top, ma qui elenchiamo sistemi completi pronti. Ad es. festival e flite li omettiamo perché qualitativamente out-of-game.*)

## 9.4 Top 10 – Voice Cloning Ability

1. **Coqui XTTS v2** – *King of zero-shot cloning.* Con solo pochi secondi di audio produce la voce target con alta fedeltà [19] [2]. Eccelle in conservare timbro e anche parte dello stile emotivo. Cross-lingual cloning notevole: può usare voce inglese e parlare italiano mantenendo il "colour" vocale. L'abbiamo visto replicare voci famose (test di community clonano attori celebri in 3s audio e suona come loro in altre frasi). Imbattibile in open-source attuale.

2. **Resemble Chatterbox** – *Pro-level cloning.* Resemble come azienda era già nota per clonazione (il loro servizio commercial Resemble AI). Chatterbox porta quell'esperienza open: con 5-10s di sample, la voce risulta immediatamente riconoscibile. In alcuni sample comparativi, la clonazione è cosi accurata che in un test AB su reddit l'88% scambiava la voce generata per l'originale [92]. Ottimo per clonare voci di attori o brand voice.

3. **Zyphra Zonos** – *Unlimited voice cloning con qualità emergente.* Zonos permette clonazione *illimitata* senza overhead di training, come Coqui. I risultati su voci inglesi presentati da Zyphra mostrano somiglianze forti (e.g. clonazione di celebrità in demo su voca.ro). Per italiano,

leggermente dietro Coqui (che ha dataset multi-ling più consolidato). Ma la tecnologia sottostante (audio-aligned encoder + SSM) è avanzata e riteniamo potrà eguagliare i primi due presto. Plus: consente *voice mixing* (cosa non comune: genera una voce intermedia tra 2 reference, utile per creare voci sintetiche originali combinando timbri reali).

4. **OpenVoice v2** – *Cross-language & style clone.* Punti forti: clone timbro con uno snippet e può farlo parlare in qualunque lingua (neanche VALL-E spinto così in open). In clone puro timbro, leggermente meno accurato di Coqui, ma la flessibilità multi-ling è unica. Utile se voglio replicare voce di persona che parla solo inglese e usarla per italiano (Coqui lo fa bene, OpenVoice è stato progettato specificamente su quell'idea [93]).

5. **Tortoise-TTS** – *Few-shot ultra quality clone.* Tortoise non è zero-shot (vuole ~1min di audio), ma se glieli dai, la somiglianza vocale è altissima. Nella comunità molti lo usano per generare doppiaggi con voce di attori (in inglese). In italiano, se addestrato, potrebbe replicare quell'accuratezza (limite: dataset ENG). Il vantaggio è che imita molto anche la *personalità vocale* (pauses, quirks), perché analizza più audio lungo. Quindi per clonare fedelmente un individuo con tutte le sfumature, Tortoise fine-tunato rimane super (il costo è lentissimo, per off-line usage come generazione messaggi voc.

6. **Bark** – *Style mimic partial cloning.* Bark come detto può replicare timbro e mood di un clip, ma non è garantito su testo arbitrario. Comunque, con un sample sufficientemente lungo, riproduce l'impronta vocale riconoscibile. Lo posizioniamo qui perché e` l'unico in grado di trasferire anche *effetti audio contestuali* (es. rumore di fondo, eco del microfono) insieme alla voce – se uno volesse clonare scenario (tipo suonare come un messaggio lasciato in segreteria con quell'atmosfera), Bark lo fa. Non esattamente use-case telefonico tipico, ma interessante.

7. **YourTTS (2022)** – Va menzionato: è stato uno dei primi modelli open multi-lingua con voice cloning, creato dal team di Glow-TTS e Soares. Può clonare voci di portoghese, inglese, francese con un embedding speaker. Rilasciato su HF (cshulby/YourTTS). Risultati: clonazione buona ma non ai livelli di Coqui v2 (che è successore evoluto). Comunque, se integrato, fornisce clonazione con architettura VITS e HiFiGAN efficiente.

8. **So-VITS-SVC & RVC** – *Retrieval voice conversion.* Non TTS di per sé, ma per clonazione vocale (specie canto) sono mainstream. Le poniamo qui per completezza: RVC e SoVITS v4 sono capaci di conversione in real-time (40ms latency con GPU per streaming di canto). Quindi *volendo* integrarle: l'utente parla -> TTS generico produce audio -> RVC trasforma in voce target al volo -> quell'audio va in chiamata. Questo pipeline clonerebbe la voce con latenza minima (<200ms). Nessun modello TTS end-to-end fa clonazione in streaming cosi – è un hack, ma RVC è pensato per streaming (già usato per voice changer su vtuber etc.). Quindi va citato: come clonazione "con poco audio" RVC batte molti (bastano 2 min audio target, produce clone convincente).

9. **ElevenLabs (closed)** – Non open, ma come riferimento: clonazione 0-shot superba. Lo citiamo solo per dire che i primi 3 open (Coqui, Chatterbox, Zonos) sono arrivati quasi a quell livello (ElevenLabs fu finora gold standard clonazione). Questo indica il trend: open colma gap, forse supererà closed presto.

10. **Google Voice Cloning (research)** – Interessa menzionare che Google & Microsoft hanno tech ancora non pubblica (ex. *AudioLM voice imitation*), che in test interni clonano voci con <1s audio (spaventoso). Nulla di open ancora, ma la top10 clonazione verrà rivoluzionata se e quando

rilasciano (magari con restrizioni). Per ora, la nostra classifica di open e modelli praticabili è come sopra.

*(Clonazione è campo in rapidissima evoluzione, la classifica 2025 potrebbe cambiare in 2026 con arrivo di Voicebox (Meta) open etc.)*

### 9.5 Top 10 – Facilità d'Uso (Deployment)

1. **Piper TTS** – *Easiest.* Un singolo comando per install, con binari precompilati e GUI di test (PiperUI). Non richiede conoscenza AI: può essere usato da chiunque sappia eseguire un exe e inserire testo. Anche integrarlo in script è banale (shell call). Perfetto per integratori non esperti di machine learning.

2. **Bark** – *Plug & Play.* `pip install bark`, e poi uno fa `generate_audio("Ciao")`. Nessuna configurazione complessa. Documentazione concisa con esempi. Non servono GPU (funziona su CPU, benché lentamente). Buono per prototipi.

3. **eSpeak NG** – *Old but gold in ease.* apt-get install espeak e hai finito. Lo includiamo perché quell'unica riga dà subito sintesi (per quanto robotica). E integrarlo in dialplan è solo chiamare espeak con text via system call. Zero complessità.

4. **Resemble Chatterbox** – *One-line via HF inference.* Su HF Space c'è demoweb, e tramite Transformers API si può usare con 5 righe. Un po' tricky come detto per controllare parametri di clonazione, ma base usage facile. Inoltre licenza MIT = niente scartoffie.

5. **Coqui TTS** – *Relativamente semplice.* Anche qui pip install e usare TTS.from_pretrained modello. Offrono anche un colab. L'unica ragione non top3 è peso download e possibili conflitti PyTorch/CUDA che newbies incontrano. Comunque, la comunità è così ampia che molte soluzioni Q&A esistono.

6. **Silero** – *Library and examples.* Se ci fosse italiano, Silero ha i migliori esempi di come integrare TTS in 10 righe di Python su qualsiasi device (snakers4 repo è pieno di snippet). Quindi per altre lingue è top, per ita non c'è modello pronto purtroppo.

7. **Mimic3** – *User-friendly packaging.* Mycroft aveva in mente utonti: mimic3 viene con un `.exe` che lancia un server web di TTS. Lato integratore, si può fare richieste HTTP e ricevere audio. Documentazione in linguaggio semplice. Purtroppo qualità modesta ma focus è facilità.

8. **Parler TTS** – *HF Transformers support.* Chi è già familiare con Transformers troverà facilissimo: è come usare un modello GPT, passi prompt e get output audio. Per novices, potrebbe servire studiare quell'esempio sul model card, ma c'è.

9. **Azzurra-voice** – *Approccio dev-savvy.* Non immediato come pip, ma forniscono quell'implementazione Rust `csm.rs` che è molto semplice da usare da CLI. Quindi se uno segue quell'approccio: `csm.rs --model azzurra --text "..."`, in 5 min di setup generi audio. Un integratore che non vuole entrare in Python e GPU stuff, può usare il binario quantizzato. Non lo mettiamo più su solo perché quell'utilità non è ancora molto conosciuta e la licenza NC può scoraggiare.

10. **Zonos** – *Installers e UI.* Zyphra ha fatto un exe di 1-click e persino mod integrato in ComfyUI per pipeline nodi. Questo dimostra focus su user experience. Punti da migliorare: per uso programmato bisogna ancora scrivere codice attorno (non c'è libreria pip). Essendo beta, mancano guide mainstream come stackoverflow threads. Ma sta migliorando attivamente.

*(Abbiamo escluso modelli come VALL-E (non pubblici) e festival (facile come apt-get ma obsoleto).)*

### 9.6 Best Overall – Modelli Consigliati per Chiamate Italiane

Tenendo conto di *tutti* i fattori (qualità voce italiana, prestazioni, funzionalità voice cloning, semplicità, supporto), presentiamo le nostre raccomandazioni principali:

**#1 Raccomandazione Principale:** `Resemble Chatterbox`
- **Modello:** *Chatterbox (Multilingual, Llama 0.5B backbone)* di Resemble AI.
- **Perché:** Offre il miglior equilibrio fra qualità vocale elevata, supporto nativo per italiano, velocità sufficiente per conversazioni in tempo reale e facilità di deployment. La voce suona naturale (quasi livello umana) [6] [7], con intonazione configurabile. La clonazione vocale integrata consente di personalizzare la voce dell'assistente (es. usare la voce di un operatore reale) con pochi secondi di registrazione. Inoltre è open-source (MIT) e ampiamente testato, riducendo i rischi di bug critici.
- **Setup:** Installare PyTorch e Transformers HF. Scaricare il modello `resemble-ai/chatterbox` da HuggingFace (circa 2GB). In uno script Python, caricare con `AutoModelForCausalLM` e `AutoTokenizer`. Per l'uso nel dialplan, consigliamo di implementare un microservizio (Flask API): la richiesta contiene il testo e un ID parlante, il servizio genera audio WAV e lo restituisce. Prevedere caching se molte richieste ripetono frasi comuni. Il tutto su un server con GPU 4GB+ (preferibilmente 8GB per margine). Impostare il servizio in start automatico sul server.
- **Performance Attesa:** Con una RTX 3050, latenza ~1 secondo per frase di 15 parole. Con RTX 3080, latenza scende a ~0.3-0.5s. TTFB può essere <300ms usando generazione streaming. Per 20 chiamate simultanee, raccomandiamo una RTX 3080 o 3090 dedicata per TTS (o due 3050 in parallelo con carico bilanciato, se l'applicazione consente splitting – meno ideale). In caso di saturazione GPU, Chatterbox fallback su CPU (con quantizzazione) produce audio in ~2-3s – ancora accettabile per risposte IVR leggermente asincrone.
- **Costo:** Stimando un server con RTX 3080 (€800) e CPU decente (€1000), ammortizzati su 3 anni = ~€50/mese. Elettricità ~€50-80/mese (se abbastanza utilizzato). Totale ~€130/mese. Per confronto, cloud TTS (Polly/Google) su stesso volume costerebbe qualche migliaio come mostrato. ROI eccezionale in scenario intensivo (oltre 90% risparmio).
*(Valore aggiunto:* Nessun costo per clonare voce, che invece servizi come Amazon Lex farebbero pagare come "Neural Brand Voice" con contratti enterprise).

**#2 Alternativa Solida:** `Coqui TTS XTTS v2` **( )**
- **Modello:** *XTTS v2 (multilingual)* dalla libreria Coqui TTS [19] [2].
- **Perché:** Offre qualità vocale comparabile al #1, con in più una capacità di clonazione vocale cross-lingual tra le migliori (basta un sample 3s per replicare voce) [32] [94]. Ideale se la personalizzazione vocale è prioritaria (es. *voce brand*). L'italiano è ben supportato (tra le 17 lingue addestrate) e la voce suona naturale. La community Coqui è un altro vantaggio – aggiornamenti e fix costanti.
- **Setup:** `pip install TTS`. Poi `tts = TTS("tts_models/multilingual/multi-dataset/xtts_v2", gpu=True)`. Va scaricato il modello (fa la libreria in automatico, ~5GB). Integrare come servizio similmente a #1. Nota: il modello è pesante e lento in esecuzione, quindi attivare `compute_type="int8"` se si usa `TTS` API per ridurre VRAM e aumentare velocità. Considerare pipeline streaming: coqui consente di passare `speaker_wav` direttamente alla chiamata TTS per clonare voce (facilitando rotazione voci).
- **Performance:** Su RTX 3050, latenza ~5s frase (no streaming). Ciò è borderline per conversazioni.

*Soluzione:* predisporre output chunk: es. se l'assistente deve dire "Buongiorno [nome], come posso aiutarla oggi?", generare prima "Buongiorno [nome]" (1.5s latenza), mandarlo, e subito dopo generare "come posso aiutarla oggi?" mentre il primo pezzo è in riproduzione. Questo dimezza la percezione di attesa. In un IVR multi-turn, la macchina può in background generare possibili risposte mentre l'utente sta ancora parlando (speculazione basata su predizione di NLU). Con tali accorgimenti, Coqui è utilizzabile. Per 20 chiamate, tuttavia, la GPU 3050 saturerebbe. Bisogna usare HPC (multi GPU) se voluto in concurrency, o profilare esecuzioni su CPU con quantization (risultato ~15s per frase, poco pratico). Dunque consigliato per volumi medi o situazioni in cui possiamo tollerare qualche secondo di attesa (es. generazione di spiegazioni lunghe in call di supporto, dove l'utente aspetta volentieri risposta elaborata).
- **Costo:** Coqui è free open (MPL-like license CPML, consente commercial con poche restrizioni). Costo hardware come #1, forse maggiore se serve cluster HPC. In uno scenario con richieste non simultanee elevate (es. call dove l'assistente parla solo quando utente tace, non molte insieme), si può gestire con una sola GPU. Stimiamo costi ~€150-200/mese includendo hardware. Cloud paragonabile costerebbe ben di più su volumi alti, quindi ROI positivo oltre soglia 1000 min/mes.

**#3 Opzione Budget/Sperimentale:** `Piper TTS + RVC Voice Conversion`
- **Modello:** *Piper TTS* per generare parlato velocemente e **So-VITS or RVC** per convertire il timbro in una voce personalizzata.
- **Perché:** Questa combinazione offre un compromesso interessante: Piper fornisce la base vocale rapida (es. voce neutra femminile italiana), e un modello di voice conversion (addestrato su un target voice di interesse) trasforma ogni output nel timbro desiderato quasi in real-time. Indicato per aziende con budget limitato che però vogliono la *voce specifica* (ad esempio, clonare la voce di un particolare operatore o del testimonial aziendale) senza dover eseguire modelli neurali enormi in tempo reale. Piper su CPU regge tante chiamate, RVC su GPU leggera regge conversione streaming audio.
- **Setup:** 1) Addestrare un modello RVC/SoVITS sulla voce target: servono ~5-10 minuti di registrazione pulita della persona, e una GPU 4GB per ~1 ora. Si ottiene un file di modello ~20MB. 2) Installare Piper e predisporre vocoder a 22050 Hz PCM. 3) Pipeline: testo -> Piper genera WAV 22kHz (0.2s di elaborazione) -> passare WAV al modello RVC (Python inference con Onnx, ~ realtime speed) -> ottenere WAV timbrato -> converti a 8kHz ulaw per linea. Tutto questo può essere orchestrato in un singolo servizio Python (usando libraries like `fairseq` or `onnxruntime` for RVC). Latenza totale ~0.5s per frase breve, quindi molto efficiente.
- **Performance attesa:** Piper per 20 flussi su CPU 16-core ~50% usage, RVC su una modesta GPU (anche una GTX1650) può convertire 20 flussi se quantizzato (perché conversione è più leggera di TTS). Ci saranno un paio di frame di ritardo aggiuntivo, ma l'utente non nota più di 200ms in più. Qualità: il timbro è molto simile al target, però la prosodia resta quella un po' monotona di Piper – quindi la voce clonata suonerà chiara e giusta come timbro, ma non super espressiva. Per molti scenari IVR, questo è accettabile (tipicamente gli IVR hanno tono piatto comunque).
- **Costo:** Piper è gratuito, RVC open (Creative Commons o MIT a seconda). Hardware richiesto è il minimo: anche un server solo CPU può fare Piper, e RVC potenzialmente su CPU per 1 stream (per 20 serve GPU). Un'unica GPU modesta (€300) basta. Totale costi hardware e elettricità < €50/mese. Questa è letteralmente la soluzione più economica per avere *voce custom offline*.
- **Nota:** È un set-up un po' *artigianale*, unendo due modelli in cascata. Richiede più lavoro di integrazione e tuning (es. normalizzare volumi tra Piper e RVC). Lo consigliamo a team piccoli con ottime competenze tecniche e che puntano al risparmio massimo. Se implementato bene, può funzionare sorprendentemente bene: la fluidità di Piper + timbro clonatosi di RVC = voce decente.

*(Menzioni speciali: - Mimic3 e MaryTTS* – potrebbero essere scelte budget (girano su Raspberry Pi, costo hw bassissimo), ma la qualità è così bassa che li escludiamo dalle raccomandazioni principali, a meno di contesti ultralowcost. - *Azzurra-voice* – sarebbe top choice qualitativa, ma la licenza NC e risorse richieste la posizionano come opzione solo se l'azienda potesse contrattare licenza commerciale con Cartesia,

allora diventerebbe una #1 in qualità e i costi ancora sostenibili. - *Zonos & Orpheus* – emergenti, valutare tra 6-12 mesi quando maturano, potrebbero sorpassare Resemble/Coqui combinando qualità+efficienza.)

---

# Sezione 10: Modelli Emergenti 2024-2025

Gli ultimi 12 mesi hanno visto un flusso continuo di nuovi modelli TTS open-source. Esaminiamo brevemente le novità e cosa aspettarsi a breve termine.

## 10.1 Rilasci Recenti (ultimi 6 mesi)

- **Microsoft VibeVoice (Ago 2025):** Microsoft ha rilasciato a fine estate un modello TTS open (MIT) per generare dialoghi consistenti a lungo termine [68]. Disponibile in vari tagli: 1.5B, 7B param, e quantizzazioni. *Breakthrough:* orientamento al dialogo: mantiene lo stesso timbro su lunghe conversazioni e consente prompt testuali su scena (es. "[Scena: attesa in aeroporto]" influenza intonazione) [95]. Performance: pesante (7B) ma c'è versione 1.5B usabile. Attualmente supporta solo inglese espressamente, ma una v3 multi-lingual è vociferata. **Vale la pena testare?** Sì per applicazioni in inglese o se si vuole studiare l'approccio per futuri modelli italiani. Se dovessero rilasciare una versione multi-lingua, sarà molto interessante: un colosso come MS può fornire dataset di alta qualità. Teniamolo d'occhio. Non ancora rilevante per italiano (lo includeremmo se appare update).

- **Boson Higgs Audio V2 (Lug 2025):** Progetto di BosonAI (Cina). Rilascio open di un modello SOTA 5.77B param con caratteristiche notevoli (emozioni, multi-speaker) [69] [70]. Già discusso sopra come top quality; emergente in quanto un team cinese indipendente è riuscito a superare BigTech in alcune metriche. Presto potrebbe uscire Higgs V3 con dimensioni ridotte ma qualità invariata (tweet Boson suggerisce compressione MoE in arrivo). **Valore per IT:** se aggiungono più lingue (Higgs V2 ha 20 lingue tra cui IT?), può diventare un'opzione, benché heavy. Già testare V2 in italiano sarebbe utile per confronti qualitativi.

- **Spark-TTS (Mag 2025):** StepFun (un'azienda di Hong Kong) ha presentato Spark-TTS, un modello 1.2B con funzionalità di controllo fine dell'emozione tramite "emotion tokens". *Performance claims:* audio near-human, ottimizzato su dataset multi-affettivo cinese e inglese (per film d'animazione). Hanno un sito con demo [72] (i primi 500 can try free). Licenza Apache. **Multi-lingua?:** ad oggi CN/EN, ma open a contributi su altre lingue. *Stato:* Beta. **Vale testarlo?** Per italiano no, a meno di fine-tuning. Ma il concetto di *emocion controll* è importante – un modello emergente italiano potrebbe integrarlo (magari Azzurra v2?). SparkTTS è un segnale del trend: TTS non solo legge, ma recita. Se appare un porting italiano, test obbligato.

- **Meta Voicebox (Giugno 2023):** Non rilasciato pubblicamente per timori misuse (era in news come *"TTS generativo troppo potente"*). Ma i contributi di Voicebox (autoencoder generativo, contesto 20s, editing fine di audio) potrebbero vedere luce in progetti open. Già alcuni modelli (e.g. VibeVoice) integrano concetti simili. **Breakthrough:** Voicebox poteva generare parlato in 6 lingue con pochi secondi training per nuova voce, e fare *speech inpainting* (riempire buchi audio). Non disponibile, ma il fatto che possano esistere implementazioni open (qualche rumor su GitHub di un clone training su libri multi-ling) significa che entro 2026 avremo modelli open replicanti Voicebox.

- **ElevenLabs is training multilingual model (Nov 2025 rumor):** ElevenLabs, leader closed clonazione, potrebbe pubblicare un paper o modello distillato open (per community rep). Non confermato, ma se succede, quel modello (probabilmente 1-2B param) sarebbe immediatamente rilevante (Eleven ha dataset e know-how proprietario forti). Un competitor open come Chatterbox li sta tallonando, quindi potremmo vedere mosse.

- **F5-TTS (Apr 2024):** Ne abbiamo citato, *"Fairytaler that Fakes Fluent and Faithful speech"* [13] – modello di ricerca (335M param) combinante diffusion e flow matching. Hanno rilasciato codice e checkpoint English. Caratteristica: alta stabilità e capacità clonare emotività riducendo artefatti (il paper dice 50% meno errori su dataset test vs VITS). Sta circolando su Reddit con entusiasmo (vedi thread su F5 vs Zonos [96] [97]). **Maturità:** ancora v1, ma considerare implementarlo in pipeline fine-tune per italiano: alcuni dev indipendenti (es. user *alien79* su HF) ha fatto fine-tune di F5 su italiano CV dataset [51] chiamandolo *F5-TTS-italian*. Risultato: qualità media (limitato dai 24 ore di CV). Ma se avesse più dati, F5 arch potrebbe far bene. Quindi emergente in senso di architettura: difatti alcuni citano "F5 did perfect clone on X voice" [98] – la realtà è un po' hype, ma è nuovo e in evoluzione. Vale la pena sperimentare se si ha tempo, per possibili miglioramenti su clone/emotion rispetto ai mainstream.

## 10.2 Progetti Promettenti in Sviluppo

- **Canary (by Nari Labs):** Un indiscrezione – Nari Labs dietro *Dia TTS* (inglese dialog-oriented) sta lavorando a *"Canary"* che sarebbe la versione multi-lingua di Dia (1.6B param, Apache lic). Non ancora pubblica, ma voci su huggingface community dicono che includerà italiano (Nari Labs fu spinta dai feedback che richiedevano spagnolo e italiano). Se esce, Canary TTS potrebbe fornire un modello open conversazionale calibrato su lingue romanze. Aspettato in Q1 2026.

- **AI4Bharat IndicTTS-2:** Il team indiano che ha fatto modelli per lingue indiane (IIT Madras) sta lanciando IndicTTS-2 con supporto esteso a 40 lingue includendo alcune europee per test. Potrebbe includere italiano (non certo). Visto che il focus loro è su low-resource e modulare, un modello addestrato su quasi 100 voci multi-ling potrebbe arrivare. Sembra che la pipeline userà VITS e F0 conditioning. Se appare con licenza Apache, da valutare come potenziale robusto (anche se voci multiple, se includono un parlante italiano, quality dipenderà da quell'unico speaker, un po' come MMS).

- **Cartesia "Azzurra Conversational" (supposizione):** Data la presentazione di Azzurra come prima pietra (focus voce "calda" presentatrice), vociferano che stiano lavorando ad un modello orientato al dialogo (magari con multiple voices e adapt all gender?). Non abbiamo fonti pubbliche, ma la logica direbbe che vorranno capitalizzare su quell'engine. Forse un *Azzurra v2* con clonazione integrata e licenza commercial friendly. Se timeline è 2026, da seguire.

- **Community fine-tunes di Chatterbox:** Essendo MIT e modulare, già su reddit utenti condividono modelli fine-tunati su specifiche voci (es. *"Chatterbox model finetuned on 100h of anime voices"* ecc.). C'è uno in particolare su voci italiane? Potrebbe emergere: esempio, un utente potrebbe prendere dataset LibriVox italiano (voce di Silvia Ceccarelli, ~30h) e fine-tune Chatterbox su quella – fornendo un modello ottimizzato come voice clonable per italiani. Queste iniziative sono imprevedibili ma probabili, e potrebbero comparire su HuggingFace all'improvviso. Vale la pena monitorare la hub HF e reddit /r/TTS per "italian voice model".

### 10.3 Dalla Ricerca alla Produzione

Riguardo ai recenti articoli accademici che potrebbero presto avere implementazioni utili:

- **VALL-E X e VALL-E M (2023):** i paper Microsoft su clonazione multi-lingua (X) e su adattamento di stile (M) hanno fornito metriche fantastiche (X ha mos >4 su tutti target cross-lingua). Implementazioni open incomplete esistono, ma c'è un progetto *"NeonVALL-E"* in arrivo da community che mira a replicare VALL-E X con EnCodec open. Se esce e supporta italiano, avremo clonazione cross-lingua SOTA open. Ci aspettiamo qualcosa entro metà 2026.

- **Generative Spoken Language Modeling (GSLM):** questa linea di ricerca (da Meta e altri) elimina il bisogno di testo intermedio, modellando direttamente audio-> audio. Per ora applicazioni: translation direct speech, voice conversion senza trascrizione. Un uso futuro: un assistente telefonico potrebbe analizzare l'audio dell'utente e rispondere generando direttamente l'audio, con un modello unico, senza pipeline separate ASR+TTS. Un progetto open in questo senso è *SeamlessM4T* (Meta 2023, translator multi). Non ancora al punto di conversare autonomamente, ma la tendenza c'è. Nessun rilascio producibile ora, ma i *foundation models* audio possono in futuro replicare TTS con contesto multimodale.

- **Massive Multi Style Transfer:** Research come *"Stylized TTS via Prompting"* (papers 2024) mostrano modelli TTS che si controllano con prompt testuali *descrittivi*. Chatterbox già implementa questa idea [99] [100]. Prevediamo più modelli usciranno con quell'approccio, semplificando ottenere output emotivo senza dover fine-tunare. Un potenziale arrivo: *"StyleTTS 3"* (se Alibaba o MS proseguono quell filone), che integrerà prompt natural language ("parla con voce arrabbiata e sussurrante").

**In conclusione**, il panorama 2024-2025 vede un trend di **convergenza**: modelli TTS open sempre più capaci e simili a quelli proprietari, con enfasi su clonazione, emozione e dialogo. Questo è ottimo per chi, come il nostro scenario, vuole implementare soluzioni locali avanzate – i "pezzi" tecnici ci sono e saranno ancora migliori nei prossimi mesi.

---

## Sezione 11: Casi d'Uso Specifici

Per illustrare la scelta dei modelli in base alle esigenze, analizziamo 3 scenari tipici nel contesto chiamate telefoniche e indichiamo i modelli ottimali per ciascuno.

### 11.1 Scenario: Receptionist Virtuale (Risponditore aziendale)

**Descrizione:** Una segreteria automatica avanzata che accoglie l'utente e gestisce richieste semplici (orari ufficio, inoltro a reparti, presa messaggi). Frasi tipicamente brevi e ripetitive, ad es. *"Buongiorno, ABC Srl, come posso aiutarla?"*, *"Attenda in linea, la trasferisco"*.

**Requisiti chiave:** - **Latenza critica (<1s)** – l'utente deve sentire subito la voce appena chiama, altrimenti potrebbe riattaccare. Il menù deve essere reattivo (se l'utente sceglie un'opzione, la prossima voce deve partire istant). - **Voce aziendale clonata** – idealmente la voce dovrebbe essere quella reale della receptionist o comunque una voce definita dall'azienda (coerente col brand). - **Affidabilità 24/7** – dev'essere robusto, zero crash, magari ridondante.

**Top 3 Modelli:**

- **#1: Piper TTS (voce custom via fine-tuning)** – *Motivazione:* Piper è estremamente reattivo e stabile. Si può addestrare una voce custom in pochi giorni usando la voce della receptionist umana (che legge un set di frasi). Il risultato sarà una voce monolingua piuttosto neutra ma con il timbro voluto, che Piper sintetizza istantaneamente. *Pro:* Instantaneo, leggero (gira anche su server piccolo), fallback facile (ha anche eSpeak integrato come riserva). *Contro:* La naturalezza non è top: su frasi brevi poco percepibile, però se la frase fosse lunga sarebbe un po' robotico. In reception, comunque le frasi sono standard e vanno benissimo.

- **#2: Coqui XTTS / Resemble (embedded in Asterisk via AGI)** – *Motivazione:* Se si vuole voce clonata di altissima qualità e abbiamo GPU a disposizione, si può integrare un clone con Coqui o Resemble. Ad es. la receptionist incide 5 frasi, il modello clona la voce e la usa per generare *tutte* le possibili risposte (ore, trasferimenti, etc.) on the fly. In pratica fungerebbe come generatore vocale che appare identico alla persona reale. *Pro:* Qualità molto alta – il cliente crederà di parlare con la segretaria di sempre. *Contro:* Latenza iniziale un po' più alta (diciamo 0.5-1s alla risposta, ancora tollerabile). Complessità maggiore: serve GPU e un po' di caching (meglio pre-generare i prompt fissi all'avvio e tenerli in RAM per playback immediato). Comunque fattibile e spesso adottato da grandi aziende (ElevenLabs offre proprio clonazione receptionist per IVR come servizio).

- **#3: Respeaker + eSpeak fallback** – *Motivazione:* In ottica affidabilità, potremmo implementare un doppio-livello: un servizio primario (es. Coqui o Resemble clonati) e un fallback come eSpeak nel caso improbabile il primo fallisca. eSpeak pronuncia perfettamente info critiche come numeri (con voce robotica). *Pro:* Redundancy – se GPU down, eSpeak entra e almeno dà il messaggio. *Contro:* Qualità scarsa fallback. In realtà, in scenario receptionist, i prompt sono sempre gli stessi: conviene generare e memorizzare gli audio in anticipo con il motore TTS scelto, e usare quelli come fallback (file statici). Così anche se il TTS runtime fallisse, hai i file salvati. Ciò può rendere inutile eSpeak. Menzioniamo eSpeak solo come ultime risorse.

**Conclusione scenario 11.1:** Il receptionist virtuale punta su affidabilità e velocità: *Piper TTS con voce custom* appare la scelta più pragmatica (nessun GPU necessario, latenza praticamente zero). Se la qualità vocale di Piper (neutra/meccanica) è considerata sufficiente dall'azienda, andrei con Piper. Se l'azienda tiene molto all'immagine e vuole la *stessa voce umana di prima*, investirei in clonazione con Coqui/Resemble, assicurandomi di gestire caching e fallback. Entrambe le soluzioni sono implementabili on-prem con costi contenuti e offrono <1s al primo audio.

## 11.2 Scenario: Assistente Clienti (Conversazioni lunghe e complesse)

**Descrizione:** Un assistente virtuale che sostiene conversazioni più estese con i clienti (es. supporto tecnico di 1° livello, prenotazione servizi, raccolta feedback). Qui l'interazione non è solo menu a scelta, ma un vero dialogo multi-turn, potenzialmente variabile e con risposte lunghe da parte del bot.

**Requisiti chiave:** - **Naturalezza ed espressività molto importanti:** l'utente deve sentirsi a suo agio a parlare a lungo, la voce deve essere empatica, non monotona. Serve modulare il tono (es. più rassicurante se cliente è frustrato, ecc.). - **Varietà di frasi:** l'assistente dirà molte cose diverse (non solo frasi pre-programmate). Quindi il TTS deve cavarsela con qualsiasi input generato dall'NLP, compresi nomi, cifre, ecc. (buona generalizzazione). - **Emotion control:** se integrato con sentiment analysis, sarebbe utile far reagire la voce (es. cliente arrabbiato -> voce calma e pacata che trasmette comprensione).

**Top 3 modelli:**

- **#1: Resemble Chatterbox (con prompt emotivi)** – *Motivazione:* Ha la miglior combinazione di qualità e flessibilità per conversazioni lunghe. Permette di inserire tag di espressione nel prompt [5], ad esempio se l'NLP rileva che l'utente è confuso, possiamo dire all'output TTS di essere particolarmente chiaro e lento. L'audio prodotto è molto naturale anche su paragrafi lunghi (Chatterbox è testato su audiolibri / script). *Pro:* Multi-turn consistency (il timbro resta invariato su conversazioni lunghe), bassa occorrenza di errori prosodici (non spezza frasi a caso, ecc.). *Contro:* Necessita di GPU decente per generare risposte lunghe senza troppo ritardo (magari RTX 3090 per generare 5s audio in 1s). Vale investire quell'hardware per un'esperienza utente significativamente migliore.

- **#2: Zonos v0.1 (dialog-oriented)** – *Motivazione:* Zonos è proprio pensato per conversazioni con contesto (in grado di gestire prompt come [sighs] e farlo). Essendo emergente, la qualità in italiano non è ancora top, ma in scenario supporto, i clienti potrebbero apprezzare la *espressività contestuale* di Zonos (fa ridere se racconto barzelletta, etc., a quanto afferma Boson). Se entriamo in 2026, ipotizziamo Zonos v0.2 con italiano migliorato – potrebbe superare Resemble su emozionalità. *Pro:* Pianificato per streaming realtime, supporta "scene prompting" (impostare all'inizio uno stile e mantenerlo). *Contro:* Beta, possibili glitch, e italiano ad ora secondario (forse risolto entro quell'anno). Lo terrei come seconda opzione se la dimensione emotiva è cruciale: si può affiancare a Resemble e testare quale fornisce conversazioni più coinvolgenti.

- **#3: Azzurra-voice + prosody modulation** – *Motivazione:* Azzurra ha la voce più calda e umana in italiano, quindi in conversazioni lunghe sarebbe quella che meno stanca l'ascoltatore. Al momento manca il controllo emotivo fine, ma con un po' di hacking (variare punteggiatura per forzare intonazioni, etc.) si può modulare. *Pro:* Qualità altissima, sarebbe come avere un operatore umano gentile dall'inizio alla fine. La voce modulata su accenti italiani genuini potrebbe creare empatia con l'utente. *Contro:* Non real-time su hardware modesto – per conversazioni lunghe serve cluster GPU. E licenza NC (impossibile usarla commercial a meno di accordi). Quindi piuttosto teorico se rimane NC. Ma se quell'assistente fosse interno (non offerto come servizio a terzi, quindi non commerciale strettamente?), arguibile da policy. In sintesi, se un'azienda è disposta a investire risorse e possibili licenze, Azzurra in un contact center ha potenziale di mantenere al telefono i clienti più a lungo soddisfatti (voce suadente e professionale).

**Conclusione scenario 11.2:** Per un assistente clienti, **consigliamo Resemble Chatterbox come prima scelta**, per la sua naturalezza e per la possibilità di integrarsi con i modelli di lingua (fornendo magari condizionamento di intensità via prompt). Su una buona GPU, potrà generare risposte di diverse righe in 1-2 secondi, che è accettabile in conversazione (il cliente magari aspetta un attimo, come se l'operatore stesse consultando una pratica). La varietà di espressioni con Chatterbox può essere gestita definendo qualche regola: es. se la risposta contiene "mi dispiace", potremmo aggiungere un tag di tono empatico. Un bel vantaggio di modelli open: possiamo *post-processare il testo di output per aggiungere indicatori di espressività*, cosa che con servizi cloud rigidi non faremmo. Seconda opzione, **Zonos** è da monitorare: se la qualità italiana sale a livello 8/10, il suo imprinting dialogico ed efficienza streaming lo renderebbero una piattaforma ideale (soprattutto se includeranno clonazione voci, utile se in call center si vuole replicare voci di diversi agenti per personalizzare).

## 11.3 Scenario: IVR Multi-livello (Call center con menu complessi)

**Descrizione:** Un sistema IVR con menu ad albero (es. “Premi 1 per assistenza tecnica, 2 per commerciale, …”, e sottomenu ulteriori, etc.). L’IVR può dover leggere prompt multipli in sequenza mentre l’utente naviga. Tipicamente usato in call center di medie/grosse dimensioni.

**Requisiti chiave:** - **Chiarezza massima:** la voce deve essere estremamente chiara e scandire bene opzioni e cifre (spesso utenti ascoltano distrattamente queste opzioni). Quindi priorità alla dizione perfetta > naturalezza emotiva. - **Low resource usage:** Questi menu sono su tutte le chiamate, e a volte in parallelo su centinaia di linee, quindi il TTS deve poter servire contemporaneità elevata senza problemi. Spesso inoltre i menu testuali cambiano raramente, quindi c’è opportunità di caching. - **Scalabilità e modularità:** Un IVR complesso può avere decine di prompt diversi (“Benvenuto…”, “Se vuoi X, premi 5”, “Grazie, ti passo Tizio…”), quindi il sistema TTS deve integrarsi bene con un database di prompt e generare all’occorrenza. Possibile pre-caricare in RAM i file audio di tutti i prompt.

**Top 3 modelli:**

- **#1: Piper TTS (multi-voice)** – *Motivazione:* Piper torna qui perché è perfetto per leggere menu: pronuncia le opzioni in modo chiaro, letterale (usa espeak-ng per G2P italiano, che è robusto su numeri e abbreviations). Non ha “personalità”, ma in un IVR ciò non serve; anzi, una voce leggermente robotica viene percepita come la classica voce telefonica – gli utenti sono abituati e la trovano comprensibile. Con Piper si possono generare tutti i prompt offline e salvarli su disco. *Pro:* Chiarezza top (nessun errore di pronuncia di voci internazionali che magari sbagliano un nome di reparto), ultra efficiente (può girare su CPU del PBX stesso se supporta moduli TTS). *Contro:* Voce monotona, ma in IVR va bene purché comprensibile. Uno potrebbe persino considerare di usare voci diverse di Piper per livelli diversi (es. voce maschile per menu principale, femminile per sottomenu, per aiutare differenziazione) – Piper ha voci in italiano per farlo.

- **#2: Coqui TTS (MozillaTTS) con caching** – *Motivazione:* Se si desidera un tocco di naturalezza in più nei menu (magari l’azienda vuole una voce brand female realistica), si può usare Coqui XTTS (o un modello multi-speaker di Coqui se clonata). Ma generare in tempo reale su ogni chiamata sarebbe inefficiente. La strategia qui: pre-generare ogni prompt IVR con Coqui e memorizzarlo come file audio (praticamente come se avessimo registrato una doppiatrice umana). A runtime, nessuna latenza perché si riproduce il file. L’unico sforzo TTS è offline quando l’albero viene creato o modificato. *Pro:* Qualità vocale altissima per i menu (l’utente potrebbe non distinguere da una persona). *Contro:* Perde flessibilità runtime – però nei menu la flessibilità non serve molto, i testi sono statici. Quindi è un approccio molto valido. In pratica si utilizza il TTS open come un “voice talent virtuale” per incidere l’IVR. Questo scenario in realtà sta già avvenendo: molte aziende usano servizi come Amazon Polly per generare prompt IVR statici invece di pagare doppiatori. Con Coqui lo fai gratis in-house, con clonazione potresti replicare la voce del doppiatore originale se serve mantenere stessa voce di prima.

- **#3: Flite TTS (libro Voice)** – *Motivazione:* Flite è un motore TTS ultra leggero derivato da Festival, con supporto basic per italiano (c’è una voce “Italian” HMM). In contesti con risorse estremamente limitate (vecchi centralini embedded, hardware dedicato con CPU scarsa), Flite può generare menu velocemente. *Pro:* integrabile in C nel PBX, scarsa latenza, zero overhead di dipendenze. *Contro:* Qualità vocale mediocre (meglio di espeak leggermente, ma monotona e percepibilmente sintetica). Lo includiamo pensando a quei sistemi legacy che magari non possono far girare PyTorch: Flite funziona su qualunque apparecchietto. Nel 2025 però è

rarissimo non poter avere almeno una Raspberry Pi – e Piper girerebbe su quell Pi con qualità migliore. Quindi Flite è ultima risorsa giusto per quell'1% di casi.

**Conclusione scenario 11.3:** Per un IVR multi-livello la scelta ottimale ricade su **Piper TTS** per esecuzione diretta (può anche essere integrato come modulo Asterisk ARI). Genera tutti i prompt con ottima chiarezza e consuma pochissimo CPU, permettendo di gestire 100 linee con un normal server. In fase di implementazione, raccomandiamo di generare le stringhe una volta e salvarle, cosicché la latenza durante la chiamata sia zero (già in PCM pronto). I *Top 2 benefici* di Piper in questo scenario: pronuncia robusta di combinazioni alfanumeriche (tipico in menu aziendali con reparti codificati, etc.), e semplicità di aggiornamento – se il testo di un prompt cambia, rigeneri quell'audio in 0.2s e lo sostituisci. Come alternative, se l'azienda vuole migliorare un IVR esistente con suoni più naturali, l'approccio di **usare un TTS avanzato offline** come Coqui per rifare tutti i file di menu è pragmatico: di fatto abbatti costi di studio di registrazione e ottieni voci coerenti. Sarà quasi invisibile all'utente finale che è TTS e non registrazione, perché non c'è latenza e suono è pulito. Dunque i modelli neurali open possono servire anche in *modalità offline batch* molto efficacemente. L'elemento *"accents regionali"* qui non conta, meglio una dizione standard neutra per tutti – Piper va benone.

---

# Sezione 12: Community & Supporto

Quando si adotta un motore TTS open-source, è utile valutare la dimensione e vitalità della community intorno al progetto e se esiste supporto locale (in italiano). Anche la probabilità che il progetto sia mantenuto a lungo termine è importante (nessuno vuole basare un sistema su tech abbandonato l'anno dopo). Valutiamo alcuni modelli chiave sotto questi aspetti.

## 12.1 Dimensione Community e Contributori

- **Coqui TTS:** Deriva da Mozilla TTS, ha una grande community internazionale. Su GitHub conta 38k⭐ e ~100 contributori nel tempo [26]. Il Discord Coqui ha oltre 1500 membri. Problemi su GitHub ottengono risposte in giorni grazie a dev volontari. C'è anche u/trysem e altri su reddit che mantengono liste e aiutano utenti [101]. Nonostante la compagnia Coqui stessa abbia cambiato focus (forse ridotto staff open), la community di utenti e contributori è tale che il progetto rimane vivo. Sviluppi come XTTS v2 e oltre potrebbero venire da contributi open (es. esiste un *fork neonbjb di Coqui* in lavorazione, come quell'utente rif nella Q/A, segno di forking attivo). Voto viabilità: alto.

- **Resemble Chatterbox:** Essendo relativamente nuovo (open da pochi mesi), sta raccogliendo rapidamente follower (già 14k⭐). Molti user su reddit hanno iniziato ad usarlo e condivider trucchi. ResembleAI l'ha rilasciato, quindi suppongo mantengano almeno un paio di ingegneri a correggere bug e accettare PR (ho visto ~43 PR aperti, segno di contributi già arrivati in 5 mesi). Essendo MIT, attrae contributori (nessuna restrizione). Ha integratori professionali (es. blog Modal, inference company, l'hanno già incluso in pipeline cloud [39]). Quindi direi community medie dimensioni ma molto attiva – e destinata a crescere perché è modello di riferimento HF trending.

- **Zyphra Zonos:** Progetto di startup piccola – community iniziale su reddit (discussione Locallama, link su HN). Hanno ~5.7k⭐ e <10 contributori (interna perlopiù) [102]. Stanno però coinvolgendo testers su Discord e integrando suggerimenti (es. la release quant a 4 bit fatta su richieste community [103] [104]). Essendo Apache, se mai Zyphra chiudesse, la community potrebbe

continuare (ma modelli 1.6B open non mancano di sostenitori se validi). Diciamo *community emergente*, piccolo numero ma entusiasti. Va monitorata la loro attività nel 2026 per conferma.

- **Mimic 3 / Mycroft:** MycroftAI purtroppo ha avuto difficoltà (quasi chiusura nel 2022). La community open source però ha *forkato* e continua come OVOS. Mimic3 è mantenuto dalla comunità (release 2023). Contributors una decina (molti ex Mycroft dev). Non enorme, ma essendo integrato in progetti voice assistant, ha uno zoccolo. Risk abbandono c'è se interest cala. In Italia la comunità Mycroft era minuscola, quindi doc in italiano zero. Tuttavia c'è un utente italiano su forum Mycroft che ha contribuito voce italiana, quindi c'è qualcuno.

- **Meta MMS:** Non esattamente community-driven – rilasciato su HF, modelli fermi all'iniziale. Essendo research by Meta, non contare su updates frequenti (non ne hanno bisogno per scopi di ricerca oltre). Però la community open (esp. autodidatti di lingue rare) scambia info su forum (es. come migliorare pronuncia con fine-tune). L'uso mainstream è basso (qualcuno su reddit di ornitologi digitali testano Xhosa TTS, cose così). Per italiano, dubito ci sia un user base.

- **Italian communities specific:** In Italia non c'è una grande community dedicata al TTS open – la discussione avviene su forum globali. Non risultano forum italiani attivi su TTS a parte qualche thread su telegram o reddit r/Italy (non specialist). Quindi il supporto è quasi tutto in inglese. All'atto pratico, gli integratori italiani dovranno interagire con community globale (cosa comune in AI). Un punto a favore: progetti come Azzurra essendo italiani (Cartesia), offrono documentazione anche in italiano e sensibilità locale. Se quell'iniziativa cresce, potrebbe costituire un centro di competenza italiano per TTS. Al momento è agli inizi (4 contributori su HF, pochi follower).

## 12.2 Documentazione e Risorse in Italiano

Abbiamo cercato risorse localizzate in italiano per i modelli: - **Documenti ufficiali in italiano:** Rari. Azzurra blog c'è intro in IT e parametri, bel segnale [58] [57]. Coqui & Co: doc solo in inglese. I tool di sintesi old (Festival, MaryTTS) avevano manuali in italiano scritti da ricercatori/tradotti (es. manuale MaryTTS di FBK, 2010). Oggi però i modelli open evoluti non hanno pagine in italiano. - **Tutorial in italiano:** Non granché. Qualche articolo su Medium IT che spiega come usare Google TTS offline (che poi era mimic). Poca roba aggiornata. - **Community italiana:** Nessun forum dedicato TTS in italiano con massa critica. Esempio su forum HTML.it c'era un thread su come installare Festival su Asterisk anni fa, ma molto datato. - **Codice Esempio in italiano:** In alcuni progetti (Mozilla Common Voice) c'era un dataset di frasi italiane. Qualcuno su Kaggle ha contributi su TTS training con dataset IT (notebook di fine-tuning su Edoardo dataset). Non tanto specifici su modelli, ma dei gist esistono.

Insomma, la *barriera linguistica* c'è: un integratore italiano dovrà avere confidenza con risorse in inglese prevalentemente. Considerando che i modelli open si affermano, c'è opportunità per scrivere guide in italiano (questo report stesso forse funge da primo compendio).

Una nota: i modelli open come Coqui e Resemble, se portati in aziende italiane, a volte generano la necessità di supporto professionale. Non c'è esattamente un numero da chiamare come faresti con un vendor, ma per Coqui esistono partner (ex. OnPremise Solutions srl, ipotetico) che se ne intendono e possono offrire supporto a pagamento. Attualmente non ne conosco di specializzati in TTS open in Italia, però alcune società di integratori di chatbot hanno team R&D su TTS (ad es. Spitch, Almawave lavorano su ASR/TTS). Quindi se un'azienda volesse un support formale, potrebbe rivolgersi a questi integratori. Non che serva necessariamente: la robustezza di modelli come Piper fa sì che raramente c'è da risolvere bug intricati a livello codice.

## 12.3 Longevità e Rischio Abbandono

Consideriamo la *viabilità a lungo termine* dei progetti chiave: - **Coqui TTS / TTS** – Rischio abbandono *moderato* (6/10): La compagnia Coqui ha pivotato su STT (Coqui Studio). L'open TTS library potrebbe rallentare senza sponsor aziendale. Tuttavia, data la licenza MPL e l'importanza del progetto, la community potrebbe forkarlo (c'è già un fork Neon improvably). Quindi anche se Coqui inc lo lascia, il codice non muore. Inoltre, i modelli addestrati come XTTS v2 sono su HF e replicabili. Quindi il rischio è di stagnazione, più che sparizione.

- **Resemble Chatterbox** – Rischio *basso* (3/10): Resemble AI è un'azienda in crescita nel voice AI. Hanno reso open Chatterbox per posizione di mercato. Probabile continueranno a investirci (migliorare modelli, magari Chatterbox v2 in futuro). Inoltre essendo MIT, se Resemble (worst case) chiudesse, la community può prenderlo su (visto che è popolare). Finché huggingface esisterà, il modello vivrà.

- **Zonos** – Rischio *moderato* (5/10): Startup piccola, se non monetizzano potrebbero dover chiudere. Ma essendo Apache e già su HF, la community (specialmente in Cina su huggingface) potrebbe integrarlo in progetti open come "AudioLLM open". Un segnale: hanno licenza permissiva (non uno schema open-core, completamente open). Questo spesso implica o cercano vendere servizi su quell'open (like support etc.), o altruismo. Direi se entro 2026 non decollano come business, il progetto potrebbe perdersi. Tuttavia, i modelli e codice resterebbero e utenti avanzati potrebbero mantenerli (difficile far evolvere senza il core team però, data la complessità).

- **Azzurra-voice / Cartesia** – Rischio *incerto* (7/10 abbandono, *speriamo di no*): Project early stage. Cartesia è neonata startup, se non ottiene abbastanza traction in business, potrebbe non rilasciare aggiornamenti open. Hanno scelto licenza NC, quindi se loro smettono, la community non può proseguire quell'esatto modello (legale). Un segnale positivo: invitano contributi e citazioni su HF, quindi hanno mentalità open in certa misura. Dobbiamo vedere se monetizzano (forse vendendo licenze commerciali, cosa che potrebbe permettere di mantenere open base e aggiornarla). Perciò do un 7/10 di rischio che il modello attuale resti solo v1 senza update, a meno di investimenti. Augurandoci invece che fiorisca: se ad es. un partner grosso li finanzia, potrebbero rilasciare modelli emergenti open (v2, v3). Tutto da monitorare.

- **Mimic3 / OVOS** – Rischio *alto* (8/10): Con Mycroft quasi defunta e l'open source che regge un po', la velocità di sviluppo è bassa. Potrebbe essere mantenuto a vita per passione, ma dubito usciranno modelli nuovi specifici (e.g. voci italiane migliori) da quell'ambito. Quindi come soluzione a lungo termine, la vedo destinata a essere surclassata e poi dismessa.

- **MaryTTS / Festival** – Già progetti "accademici storici", essenzialmente congelati (qualche commit manutenzione su Mary fino al 2020, festival 2015). Rischio abbandono ininfluente: li consideriamo legacy.

In conclusione, i modelli più promettenti in termini di *sicurezza futura* sono quelli con licenze aperte e grandi community (Chatterbox, Coqui TTS). Il minor sicuri sono modelli one-company closed license (Azzurra NC). In un'ottica di adozione aziendale, la direzione è: preferire soluzioni con licenza permissiva e community robusta, ed eventualmente stipulare contratti di supporto con integratori se necessario (per mitigare rischi di dover risolvere bug da soli). Va detto che il codice TTS open è ormai piuttosto maturo; la maggior parte dei bug fix risolti. Rari problemi possono essere: pronuncia errata di qualche parola (risolvibile fiddlando G2P), performance tuning su architetture specifiche. Tutte cose che con un team tecnico interno si gestiscono.

Quindi sul lungo termine, **scommettere su open TTS appare relativamente sicuro**, in quanto se pure un progetto si ferma, i modelli e codici restano e un altro può prendere il testimone. Al contrario, un servizio cloud che chiude (o alza i prezzi) non lascia nulla in mano – scenario che con open non succede. Questo va presentato come argomento pro per convincere stakeholder ad adottare TTS locale open: si investe ora, e la soluzione rimane per sempre a disposizione, personalizzabile anche se il vendor cessa.

---

## Sezione 13: Guida di Test Pratico

Per garantire che le scelte di modelli e configurazioni soddisfino i requisiti, è opportuno condurre dei test pratici. Forniamo linee guida e script d'esempio per testare i modelli selezionati, misurarne performance e qualità.

### 13.1 Setup Ambiente di Test

**Hardware di test:** per uniformità, predisporre una macchina con RTX 3050 Ti (4GB) e CPU 8 core, 32GB RAM. Sistema Linux Ubuntu 22.04 (o Windows 11 con WSL2). Installare driver NVIDIA e toolkit (CUDA 11.x).

**Software e modelli da installare:** - Python 3.10+ ambiente, installare pacchetti: `TTS` (Coqui), `transformers` (Resemble, Parler), `pip install piper-tts`, etc., a seconda di modelli testati. - Scaricare i modelli pre-addestrati necessari (coqui xtts, chatterbox, etc.) in cache locale prima del test per evitare tempi di download durante test.

**Script generico di test (pseudo-Python)**:

```
# test_tts_model.py - Template
import time
import soundfile as sf
# This example uses Coqui TTS, adapt for each model accordingly
from TTS.api import TTS

# Setup model
model_name = "tts_models/multilingual/multi-dataset/xtts_v2"  # modify for
each model
tts = TTS(model_name, gpu=True)

# Phrases to test:
phrases = [
    "Buongiorno, come posso aiutarla?",  # short greeting
    "Il suo ordine numero 12345 è stato spedito e arriverà domani.",

"Gentile cliente, la informiamo che per parlare con un operatore resti in
linea.",
    "Premi 1 per supporto tecnico, 2 per informazioni commerciali.",
    "Mi dispiace, potresti ripetere il codice per favore?"  # emotive
scenario
]
# Optionally test voice cloning:
```

```
speaker_wav = "sample_reference.wav"  # path to 5s of target voice (if
supported by model)

# Warmup
_ = tts.tts("Test", speaker_wav=speaker_wav if 'speaker_wav' in
tts.tts.__code__.co_varnames else None)

# Loop through phrases
for text in phrases:
    start = time.time()
    if 'speaker_wav' in tts.tts.__code__.co_varnames:
        # If model supports voice cloning reference
        audio = tts.tts(text, speaker_wav=speaker_wav)
    else:
        audio = tts.tts(text)
    elapsed = time.time() - start
    print(f"Generated {len(audio)/tts.sampling_rate:.2f}s audio in {elapsed:.
2f}s (RTF={len(audio)/tts.sampling_rate/elapsed:.2f}x)")
    # Save output for listening
    sf.write(f"output_{model_name.split('/')[-1][:5]}
_{phrases.index(text)}.wav", audio, tts.sampling_rate)
```

Questo script esegue ogni frase, misura tempo impiegato e calcola Real-Time Factor (RTF). Salva i WAV per ascolto.

**Da replicare** per ogni modello: modificare la parte di init e sintesi (Resemble chatterbox userà Transformers, Piper userà `piper_tts` con pipeline differente, eSpeak possiamo chiamare subprocess).

**Esempi di come variare:** - Per *Resemble chatterbox*: usare `from transformers import AutoModelForCausalLM, AutoTokenizer` ecc. - Per *Piper*: usare `import subprocess` con chiamata a binario `piper` CLI per misurare tempo end-to-end (oppure usare il python wrapper). - Per *eSpeak*: `subprocess.run(["espeak-ng", "-v it", text])` redirigere out in WAV via param `-w output.wav`.

### 13.2 Test Benchmark Italiano

Abbiamo definito in sezione 3 alcune frasi standard. Riepiloghiamo e spieghiamo: 1. **Breve saluto semplice:** "Buongiorno, come posso aiutarla?" – serve a notare pronuncia di “Buongiorno” (verifica intonazione, u aperta/chiusa) e fluidità sul punto interrogativo (ci aspettiamo sale di tono). 2. **Frase media con numeri:** "Il suo ordine numero 12345 è stato spedito e arriverà domani." – qui verifichiamo come viene letto *12345* (coerente "dodicimilatrecento..."). Controlliamo cadenza su frase affermativa neutra. 3. **Frase lunga e formale:** (complessa con subordinate) – utile a vedere se modelli mantengono coerenza su frase articolata e corretta enfasi. I modelli peggiori faranno pause strane o monotonia, quelli buoni useranno virgole come pause e cambieranno tono su fine periodo. 4. **Frase con comandi e enumerazioni:** "Premi 1 per..., 2 per..." – simula menu IVR. Vogliamo vedere se i modelli mettono micro-pause tra "Premi 1" e "per supporto...". E pronuncia di "1" vs "uno" (in un contesto di prompt, alcuni TTS direbbero "uno", ma l'usabilità in IVR preferisce “Premi uno”. Comunque “Premi uno” va bene). 5. **Frase con scuse e domanda:** "Gentile signor Bianchi, la contatto da Milano." e "Mi dispiace, ... per favore?" – frasi con nome proprio (test accentazione su Bianchi, e pronuncia di Milano: controlliamo se modelli

english-trained fanno errori come pronunciare "Milàno" sbagliato – improbable se data italian support, ma un test). Il "Mi dispiace, potresti ripetere..." serve a vedere se modelli esprimono un po' di contrizione (es. un calo tonale su "Mi dispiace").

**Standard di valutazione soggettiva:** come se facessimo un MOS test con 5 stelline: - Indistinguibile da umano -> 5. - Leggermente artificiale ma naturale -> 4. - Comprensibile ma robotico -> 3. - Difficile da capire -> 2. - Inaccettabile -> 1.

Durante i test, ascoltare con cuffie e anche su altoparlante di telefono (simulate degrade, compress audio 8k ulaw e riascoltare – per replicare reale percezione utente). Raccogliere le osservazioni come fatto in sez 3: pronuncia corretta? intonazione interrogativa? pause ai punti giusti? E eventuali errori (es. se pronuncia "12345" digit per digit o erroneamente "dodici tre quarantacinque"?).

**Raccomandazione:** Far valutare i file a 2-3 ascoltatori (colleghi) senza dire quale modello, e far dare punteggio. A volte lo sviluppatore è influenzato dal conoscendo del modello – un test cieco aiuta.

## 13.3 Metriche da Misurare

Per ogni modello testato, raccogliamo:

- **Cold start time:** tempo che il modello impiega a iniziare la prima inferenza. (Può includere load in RAM e init GPU). Ad esempio Coqui TTS, la prima chiamata spesso compila autoregress kernel e può essere lenta. Cold start rilevante se si pensa di spin-up/spin-down servizi su richiesta.
- **Warm inference time per frase breve e lunga:** come nel script, abbiamo varie lunghezze, possiamo registrare latenza per ogni.
- **Memory peak:** controllare con `nvidia-smi` e `ps` l'uso di VRAM e RAM durante generazioni lunghe. Dare un'idea di quante istanze parallele possono coesistere.
- **GPU utilization:** se test su GPU, `nvidia-smi dmon` per vedere se la GPU è saturata al 100% o se c'è margine (alcuni modelli non utilizzano bene la GPU e sono bound da CPU; scoprire se cosi, es. eSpeak su GPU fa nulla ovvio).
- **Audio quality soggettiva:** prendere gli output generati e assegnare punteggio su intelligibilità e naturalezza. Forse misurare WER (convertire audio generato in testo con un ASR e confrontare col testo input). Un WER alto indicherebbe pronuncia erronea. Possiamo farlo: usare ad es. Google STT sugli output audio – se modelli top, l'ASR ritranscriverà perfettamente; se modelli scarsi, l'ASR avrà errori (ASR se confuso indica audio poco chiaro). Non è perfetto come MOS, ma un autometrica aggiuntiva.
- **RTF medio:** giù calcolato dall'output e tempo. Dare quell'indice (es. RTF 0.5 = 2x slower than real-time, RTF 2.0 = 2x faster).

Mettere tutto in tabellina finale per confrontare modelli.

Tali metriche guideranno la scelta: se un modello appare con RTF <0.1 su frasi lunghe, sappiamo che non regge scenario conversation realtime. Se WER dell'ASR su quell'audio è 0, quell'audio è chiaro (questo può convincere manager che l'output è affidabile).

Possiamo includere snippet audio (embedding in eventuale presentazione) per far toccare con mano la differenza vocale.

---

# Sezione 14: Limitazioni e Problemi Noti

Infine, è opportuno documentare le limitazioni specifiche di ogni modello e possibili workaround che la community ha sviluppato.

## 14.1 Problemi Tecnici Noti

- **Coqui TTS (XTTS):**
- *Bug:* mem leak GPU segnalato in versioni <0.9 (risolto 0.10).
- *Incompatibilità Windows:* di default usa espeak phonemizer che su Windows non trovava librerie – fix: installare espeak e set path.
- *Performance bug:* su GPU con VRAM <8GB, disabilitare noise conditioning (flag in config) riduce qualità ma evita out-of-memory per frasi lunghe.
- *Large text bug:* se input > 300 caratteri, a volte Coqui produce output tagliato (stop prematuro). Workaround: spezzare il testo in frasi e unirle.
- **Chatterbox Resemble:**
- *GPU required:* attualmente generare con CPU è possi bile ma lentissimo (0.1x real-time). Non esattamente bug, ma una limitazione: va considerato obbligo GPU per produzione.
- *Memory issue:* Versione initial aveva glitch su generazione >20s (peak memory usage soared e crash). Soluzione: generare max 15s alla volta.
- *HFT5 bug:* un utente segnalava che su Torch 2.0 c'era un errore in quantization (lo risolsero con patch FlashAttn2).
- *Documentation gap:* come detto, la clarifica tra `Chatterbox mo del vs HF version` confondeva (ma nel frattempo la doc migliorata).
- **Piper:**
- *Language config bug:* v0.0.2 il modello italiano aveva un JSON config errato che causava pronuncia scorretta di "j" e "z" – fix arrivato in v0.0.3.
- *Audio glitches RK3588 NPU:* (dettaglio hardware specific) su dispositivi ARM con NPU quant, alcuni notavano output con tic – risolto abbassando amplitude param.
- *Limited punctuation support:* Piper non sempre modula bene su "?" e "!" (lo sviluppatore consiglia di usare `<Q>` tag se serve, come da readme).
- **Bark:**
- *Non determinismo:* genera output leggermente diverso a ogni run, il che è un problema se vuoi replicare esattamente un prompt audio per uniformità. Workaround: fissare random seed (però funziona partial, Bark rimane stocastico).
- *Unsupported content type images:* (nel UI huggingface c'era bug con embed error, irrilevante back-end).
- *Speed:* uso intensivo CPU al 100%, c'è un fork chi ha integrato ONNX per velocizzare (non ufficiale).
- *Tune needed per efficacia italiano:* a volte Bark confonde lingue se testo ha anglicismi, workaround – forzare roman script tag. Ad es. `[it] Ciao... [it]`.

- **Vocoder standalone (HiFiGAN):**

- *Aliasing su 8kHz:* modelli HiFiGAN e Universal vocoder allenati su 22k producono artefatti se downsamplati poi. Sol: addestrare vocoder direttamente a 16k se si prevede quell'uso.

- *Onnx export issues:* alcuni vocoder (MelGAN) avevano strati non supportati da onnx opset, hamper di effic. Community fornisce patch script per conv param in conv.

- **ASR interplay:** Non strettamente TTS, ma un limite di end-to-end user experience: se utente parla velocemente, l'ASR e l'elaborazione potrebbero sovrapporsi con TTS. L'IVR deve essere in grado di *interrompere il TTS immediatamente* (barge-in). Questo non è un problema dei modelli di per sé, ma di design del sistema. Quindi la limitazione e workaround: modelli come Coqui e Resemble non fanno streaming fine, quindi l'app deve avere meccanismo per stoppare generazione in thread parallelo (es. raising exception, efficiente in Coqui – kill thread).

## 14.2 Limitazioni specifiche Italiano

- **Pronuncia di nomi stranieri:** La maggior parte dei modelli italiani pronuncia parole non italiane letteralmente con regole italiane. Esempio: *"URL"* viene letto "urla" in eSpeak; *"JSON"* come "gison". Workaround: per sigle e termini inglesi, usare manual spelling ("jason" per JSON, se voluto).
- **Accenti regionali:** Nessun modello open attualmente riproduce accenti dialettali italiani. Se l'app volesse rispondere con coloritura locale (es. un assistente napoletano per user napoletani), non c'è out-of-box. Workaround: clonare una voce dialettale (bisogna avere registrazioni con quell'accento e fine-tune). Oppure manipolare audio (pitch, etc., con risultati modesti).
- **Cantilena robotica su frasi lunghe:** Notato in modelli come GlowTTS e Tacotron: su frasi molto lunghe senza punteggiatura, perdono intonazione corretta (diventano monotoni). Soluzione: spezzare frasi lunghe inserendo virgole ogni ~15 parole (questo suggerito in forum Coqui).
- **Numeri e date:** Diversi modelli leggono i numeri una cifra alla volta. In italiano in contesti informali è preferibile dire "dodici tre quarantacinque" come numero telefonico, ma in contesti di ordini di acquisti è meglio "dodicimilatrecento45"? Inconsistenza: eSpeak direbbe "dodicimilatrecento45" (non ottimo). Silero se avesse ita, li pronuncerebbe come numero intero. Spesso la soluzione migliore è scrivere nel testo come si vuole: "12 345" come "12 3 45" per forzare lettura parted (o scrivere a lettere).
- **Acronimi e Sigle:** TTS letterali di solito pronunciano *"IBM"* come "ibiemme" correttamente o come "ibm"? Dipende se c'è lexicon. Coqui ha lexicon con marche auto, IBM ecc. se addestrato su Libri italiani, ma non garantito. Workaround: se stringa di 2-3 lettere, inserire spazi: "I B M" per farlo fare lettera per lettera. Oppure usare fonetica if model supports.
- **Anglicismi e pronuncia corretta:** Ad es. *"hardware"* viene spesso italianizzato come "arduer". Un TTS potrebbe dire "hard-ware" in due sillabe separate. Non c'è facile fix a meno di personalizzare dictionary. Coqui TTS consente di passare una pronuncia personal (you can provide ARPABET for a word). Altri come Piper/eSpeak permettono di definire alias nel voice config (es. "hardware"->"harduer").
- **Omissione di soggetto implicito:** I modelli ovviamente leggono letteralmente il testo. In italiano spesso sottintendiamo il soggetto ma scriviamo pronome per chiarire. A volte potremmo voler scrivere in input un pronome solo per modulare intonazione. Esempio: scritto "(pause) lo farò." vs "lo farò." – la resa cambia. Diventa un'arte fine mettere punteggiatura invisibile (virgole, ...).
- **Troncamento e apostrofi:** Alcuni modelli potrbbero confondersi su parole con apostrofo seguite da vocale differente da come appare. E.g. "un'idea" – eSpeak male alvevola come [un]

[idea] staccati. GlowTTS invece impara da data e pronuncia ok. Questo dipende molto da training. Workaround: se notato errori su un apostrofo specifico, scrivere per esteso (“una idea”).

- **Femminile vs Maschile TTS:** Non limitazione linguistica ma di modelli – se serve una voce femminile e il modello open ha solo voce maschile italiano (caso di eSpeak prima erano 1M e 0F), quell è un limite. Oggi: Coqui e Chatterbox possono generare entità vocali di genere differente se condizionate (coqui xtts ha param speaker embedding, c’è italiano female in training?). For fine tuning, dataset voci femminili italiana esiste (es. DAT Italian).
- **Velocità parola:** L’italiano parlato velocemente a volte un TTS non rispetta la velocità se non gliela setti. E.g. in enumerazione di menu, un umano direbbe “Premi 1 per X, 2 per Y” con un certo ritmo. Il TTS neurale a volte li pronuncia troppo lenti perché cerca di articolare perfetto. Workaround: aggiungere virgole per forzare brevità o param di tempo se disponibile. Esempio in Coqui TTS, config allow global speed factor.

Molti di questi workaround possono essere integrati in un *post-processore del testo*: prima di dare il testo al TTS, eseguire sostituzioni (numeri -> parole, sigle -> letter spacing, etc.). Questo può eliminare 90% dei problemi di pronuncia. Di fatto, spesso la pipeline in IVR prevede già “text normalization” (ad es. per dire orari in modo colloquiale). Quindi è un aspetto di implementazione più che limite insormontabile dei modelli.

## 14.3 Soluzioni e Patch dalla Community

- **Pronunciation dictionaries:** Coqui TTS consente di passare un dizionario personalizzato. La community ne ha creati per varie lingue (CEFR in Eng, in Italian qualcuno condiviso?). Forse no, ma uno può farlo. eSpeak permettte `-X` param per debug pron e definire conversioni in sorgente (impraticabile per integr).
- **Phonetic input:** Resemble Chatterbox e Parler TTS essendo basati su LLM audio, non espongono facilmente controllo fonetico. Coqui sì (puoi inserire notazioni ARPABET nel testo fra `{{ }}`). In italiano sarebbe SAMPA (if support). Insomma, se proprio un nome esce male, con Coqui e Piper puoi hack solution, con modelli black-box no (devi mis-spellare in input).
- **Community forks:**
- Un fork di Piper chiamato `OHF-Voice Piper1` integrato ONNX e quant, risolvendo licenza to GPL e aggiungendo streaming. Non mainline, ma chi vuole quell feature può usarlo (accettare lic GPL).
- Un utente (MattePalte) su reddit scrive di un suo TTS (Verbify) con UI – potrebbe includere fix come GUI per risolvere param frustration (es. slider di velocità).
- **Arxiv to code:** Molti fix emergono come implementare idee di paper. Ad es. intonation control: c’è un repository *“Coqui TTS prosody branch”* in cui un dev ha integrato un layer att ne per controllare mel contour con tags. Non ufficiale ma a portata se uno volesse quell fix.
- **Italian acronyms**: Il cheat di scrivere con punti (I.B.M. vs IBM) aiuta modelli a sillabare. Bastano find&replace regole (e la community ne ha compilate: su forum jarvis c’è un utente che ha condiviso lista di 100 acronimi italiani comuni e come dovrebbero essere pronunciati).
- **Synth voice too robotic?** – Spesso si suggerisce di aggiungere un leggero reverb o background noise per farla sembrare più naturale all’orecchio (il cervello percepisce voce con un po’ di noise come proveniente da un ambiente reale vs troppa pulizia suona finta). Per call center, già la linea introduce rumore, ma se volessimo calibrarlo: c’è chi passa l’audio TTS tramite un filtro band-limit G.711 di proposito e aggiunge un minuscolo static, per matchare le altre chiamate e non far spiccare come troppo “sterile”. Sono refine trick da audio engi.

**In sintesi**, ogni modello ha le sue limitazioni ma quasi sempre c’è un workaround noton (con regole testuali o param). L’unica limitazione seria senza soluzione è la *lack di emozione genuina su output se modello non la supporta.* Su questo fronte si può solo scegliere un modello che la supporta di design,

oppure attendere evoluzioni. In contesti molto critical (es. assistente psicologico?), ancora i modelli open potrebbero non trasmettere sufficiente calore.

---

**Chiudendo**, questo report dimostra che i modelli TTS open-source sono maturi per implementazioni locali in italiano, con vantaggi di costi e personalizzazione. Abbiamo esaminato decine di modelli e soluzioni, identificando i più adatti in vari scenari. Si raccomanda di iniziare con prove su piccola scala (POC) utilizzando i modelli top individuati (Resemble Chatterbox, Piper, Coqui TTS) per confermare sul campo i riscontri positivi qui raccolti, e quindi procedere a un deployment completo on-premise consapevole e calibrato sugli obiettivi e i vincoli hardware dell'organizzazione. Con l'evoluzione rapida del campo, un sistema TTS locale installato oggi potrà essere migliorato integrando nuovi modelli emergenti nei prossimi anni senza stravolgere l'infrastruttura – un altro punto a favore dell'approccio open e modulare.

**Oggi (2025)** il riferimento per qualità e flessibilità italiano è Resemble Chatterbox, ma all'orizzonte c'è una competizione vivace: Coqui, Zyphra, Cartesia e altri spingeranno i confini. Scegliendo una soluzione open-source, l'azienda mantiene la libertà di aggiornarsi col migliore modello man mano che appare, senza lock-in a un vendor: un investimento realmente *future-proof*.

---

1 3 30 31 78 79 XTTS — open-source TTS in 13 languages with voice-cloning. | by Eren Gölge | Machine Learns | Medium
https://medium.com/machine-learns/xtts-open-source-tts-in-13-languages-e05c19782a03

2 19 27 28 29 32 33 34 94 coqui/XTTS-v2 · Hugging Face
https://huggingface.co/coqui/XTTS-v2

4 14 15 16 24 25 50 68 71 72 73 74 95 101 Best local open source Text-To-Speech and Speech-To-Text? : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1f0awd6/best_local_open_source_texttospeech_and/

5 36 37 GitHub - resemble-ai/chatterbox: SoTA open-source TTS
https://github.com/resemble-ai/chatterbox

6 7 18 39 40 56 69 70 89 92 The Top Open-Source Text to Speech (TTS) Models | Modal Blog
https://modal.com/blog/open-source-tts

8 46 96 97 GPT-Sovits V3 TTS (407M) Release - 0-Shot Voice Cloning , Multi Language : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1jbyg29/gptsovits_v3_tts_407m_release_0shot_voice_cloning/

9 Streaming real-time text to speech with XTTS V2 - Baseten
https://www.baseten.co/blog/streaming-real-time-text-to-speech-with-xtts-v2/

10 22 Introducing speech-to-text, text-to-speech, and more for ... - AI at Meta
https://ai.meta.com/blog/multilingual-model-speech-recognition/

11 Matcha-TTS: A fast TTS architecture with conditional flow matching
https://arxiv.org/abs/2309.03199

12 [ICASSP 2024] Matcha-TTS: A fast TTS architecture with conditional ...
https://github.com/shivammehta25/Matcha-TTS

13 F5-TTS: A Fairytaler that Fakes Fluent and Faithful Speech with Flow ...
https://arxiv.org/abs/2410.06885

17 45 Text To Speech Open Source: 21 Best Projects 2025 Guide
https://qcall.ai/text-to-speech-open-source

20 21 41 42 43 44 80 81 86 87 93 myshell-ai/OpenVoiceV2 · Hugging Face
https://huggingface.co/myshell-ai/OpenVoiceV2

23 Text-to-Speech in Italian? · Issue #22 · Sharrnah/whispering-ui
https://github.com/Sharrnah/whispering-ui/issues/22

26 The Top 23 Speech Synthesis Open Source Projects
https://awesomeopensource.com/projects/speech-synthesis

35 A possible 4x-5x faster performance increase in coqui_tts ... - GitHub
https://github.com/oobabooga/text-generation-webui/issues/4712

38 Chatterbox - Free Open Source Text to Speech Model - Resemble AI
https://www.resemble.ai/chatterbox/

47 RVC-Boss/GPT-SoVITS: 1 min voice data can also be used ... - GitHub
https://github.com/RVC-Boss/GPT-SoVITS

48 High-Logic/Genie-TTS: GPT-SoVITS ONNX Inference Engine ...
https://github.com/High-Logic/Genie-TTS

[49] [75] [88] Fast and local open source TTS engine. 20+ languages, multiple voices. Model size 25MB to 65MB. Can train on new voices. : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1mi6brm/fast_and_local_open_source_tts_engine_20/

[51] Text-to-Speech Models - Hugging Face
https://huggingface.co/models?pipeline_tag=text-to-speech&p=2&sort=downloads&search=tts

[52] Silero V3: fast high-quality text-to-speech in 20 languages with 173 ...
https://www.reddit.com/r/hackernews/comments/vghwst/silero_v3_fast_highquality_texttospeech_in_20/

[53] [54] [55] [57] [58] [84] cartesia/azzurra-voice · Hugging Face
https://huggingface.co/cartesia/azzurra-voice

[59] [60] [61] [62] [65] [66] [67] [77] [103] [104] Zonos-v0.1 beta by Zyphra, featuring two expressive and real-time text-to-speech (TTS) models with high-fidelity voice cloning. 1.6B transformer and 1.6B hybrid under an Apache 2.0 license. : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1imdnap/zonosv01_beta_by_zyphra_featuring_two_expressive/

[63] [64] Zyphra
https://www.zyphra.com/post/beta-release-of-zonos-v0-1

[76] GitHub - huggingface/parler-tts: Inference and training library for high-quality TTS models.
https://github.com/huggingface/parler-tts

[82] [83] [90] [99] [100] parler-tts/parler-tts-mini-multilingual-v1.1 · Hugging Face
https://huggingface.co/parler-tts/parler-tts-mini-multilingual-v1.1

[85] Install Zyphra Zonos TTS in 1-Click - YouTube
https://www.youtube.com/watch?v=ZQLENKh7wIQ

[91] What is the best open source TTS model with multi language support? : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1lnejb6/what_is_the_best_open_source_tts_model_with_multi/

[98] F5-TTS! They DID IT! Perfect voice clone with Emotion with a 10 ...
https://www.youtube.com/watch?v=6i0cXSvyz98

[102] GitHub - canopyai/Orpheus-TTS: Towards Human-Sounding Speech
https://github.com/canopyai/Orpheus-TTS