1 . Gather the prerequisites

What	Why you need it	Quick check-list
Twilio project with WhatsApp Business number that already sends/receives messages	Calling simply extends the same sender.	– WABA approved– Phone number in "Online" state
WhatsApp Business Calling enabled	Turns that sender into a VoIP end-point.	From 15 July 2025 GA – no private beta token needed.
TwiML Voice Application SID	Webhook where Twilio will POST the inbound call (and where you'll start your media stream).	Create one in Console → Voice → TwiML Apps.
Server that speaks WebSockets (Node, Python, Go …)	Receives Twilio Media Streams and relays them to ElevenLabs.	Low-latency, ideally < 200 ms RTT
ElevenLabs account & API key	Gives you STT, LLM orchestration and low-latency TTS voices.	Create at https://elevenlabs.io (free tier works)

2 . Enable voice on your WhatsApp sender

```
curl -X POST \

https://messaging.twilio.com/v2/Channels/Senders/EXXXXXXXXXXXXXXXXXXXXXXXXX
\
  -u "$TWILIO_ACCOUNT_SID:$TWILIO_AUTH_TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{
        "configuration": {
          "voice_application_sid": "APXXXXXXXXXXXXXXXXXXXXXXXX"
        }
      }'
```

Setting the `voice_application_sid` activates WhatsApp Business Calling for that sender. Undo it by setting the field to `null`. (Twilio)

Heads-up: Business-initiated voice calls are *not* allowed from numbers in the U S, Nigeria, Canada, Egypt, Vietnam or Turkey (user-initiated is fine). (Twilio)

3 . Return TwiML that opens a bidirectional Media Stream

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Response>
  <Connect>
    <Stream url="wss://YOUR-SERVER.COM/twilio"
            track="inbound_track"
            bidirectional="true" />
  </Connect>
</Response>
```

- `<Connect><Stream>` is what makes the stream **bidirectional**; you can both receive the customer's audio and push your synthesized reply back down the same socket. (Twilio, Twilio)

Twilio will now fork the WhatsApp VoIP audio and start firing JSON frames like:

```
{
  "event": "media",
  "media": { "payload": "base64_pcmu" }
}
```

(reference: *Media Streams → WebSocket Messages*). (Twilio)

---

4 . Bridge Twilio ↔ ElevenLabs in real time

Below is the minimal control-flow you need to run inside your WebSocket handler:

1. **Decode Twilio audio** – inbound is 8 k Hz μ-law or PCM-16 depending on the call region.
2. **Forward each chunk to ElevenLabs STT** (`/v1/speech-to-text`) or let the ElevenLabs *Agent WebSocket* take raw PCM directly. It quickly emits transcripts. (ElevenLabs, ElevenLabs)
3. **Send the transcript to your LLM / business logic** (ElevenLabs agent can do this for you or you can plug your own).
4. **Stream the agent's text response to ElevenLabs TTS** (`/v1/text-to-speech/VOICE_ID/stream`) or the **Conversational-AI WebSocket** if you're using a full ElevenLabs agent. The endpoint returns PCM or Opus in very small fragments (< 50 ms). (ElevenLabs, ElevenLabs)
5. **Base64-encode that PCM and send a `media` frame back to Twilio** over the same socket:

```
{
  "event": "media",
  "streamSid": "MZXXXXXXXXXXXXXXXXXXXX",
  "media": { "payload": "<base64_pcm>" }
}
```

Twilio mixes the audio straight into the WhatsApp call.

Tip: If you prefer *zero glue code*, ElevenLabs now offers a **Twilio integration** screen (ConvAI → Phone Numbers → *Twilio*), where you paste your Twilio credentials and it

auto-provisions the Media Stream URL for you. (Docs: *Phone Numbers › Twilio integration*) (ElevenLabs)

---

5 . Sample Node.js skeleton

```
import WebSocket, { WebSocketServer } from 'ws';
import { streamToElevenLabs, ttsStream } from './elevenlabs.js';

const wss = new WebSocketServer({ port: 8080 });

wss.on('connection', twilioWs => {
  let elWs;

  twilioWs.on('message', async msg => {
    const frame = JSON.parse(msg);
    if (frame.event === 'start') {
      // open ElevenLabs agent socket
      elWs = await streamToElevenLabs();
    }
    if (frame.event === 'media') {
      elWs.send(frame.media.payload);          // raw PCM → STT
    }
  });

  // Relay agent responses back to Twilio
  const sendToTwilio = payload =>
    twilioWs.send(JSON.stringify({ event: 'media', media: { payload } }));

  elWs?.on('message', async agentFrame => {
    const text = JSON.parse(agentFrame).agent_response;
    for await (const chunk of ttsStream(text)) {
      sendToTwilio(chunk);                     // PCM base64
    }
  });
});
```

---

6 . Testing the whole loop

1. **Invite yourself**: From your business WhatsApp thread send a template with a `VOICE_CALL` button and tap it. (Twilio)
2. Observe in the Twilio Console → Voice → Calls that the call hits your TwiML app and the stream opens.
3. In your server logs you should see Twilio → ElevenLabs → Twilio packet flow within ~300 ms each way.
4. Tweak your *turn-taking* settings in ElevenLabs if you still talk over the customer. (ConvAI → Voice → Turn taking.)

---

7 . Cost & scaling notes

- **Twilio** bills WhatsApp voice per minute; same rate as an app-to-app call, *not* PSTN. Check the new template-based pricing for messages you still send. (Twilio Help Center)
- **ElevenLabs** Conversational-AI minutes include STT + TTS + LLM orchestration; silence > 10 s is discounted 95 %. (ElevenLabs)
- Media Streams are regioned; pick `IE1` for low latency if most users are in Europe. (Twilio)

You're live 🎉

That's all that's needed to let customers tap the regular green "call" icon in WhatsApp and talk to an ElevenLabs-powered AI agent—no browser links, no app switching, just pure WhatsApp.

Happy building, **Algominds**! If you hit any snags (e.g., voice latency > 800 ms, transcription edge-cases or compliance requirements) let me know and we can dive deeper.