# AI-Powered Call Agent

This documentation provides a comprehensive technical overview of an AI-driven telephony system built using modern cloud services and machine learning APIs. The system enables automated phone call handling with natural language processing capabilities, integrating speech recognition, text generation, and voice synthesis components. Key features include real-time conversation management, transcript storage, and scalable deployment architecture.

## System Architecture Overview

### Core Components

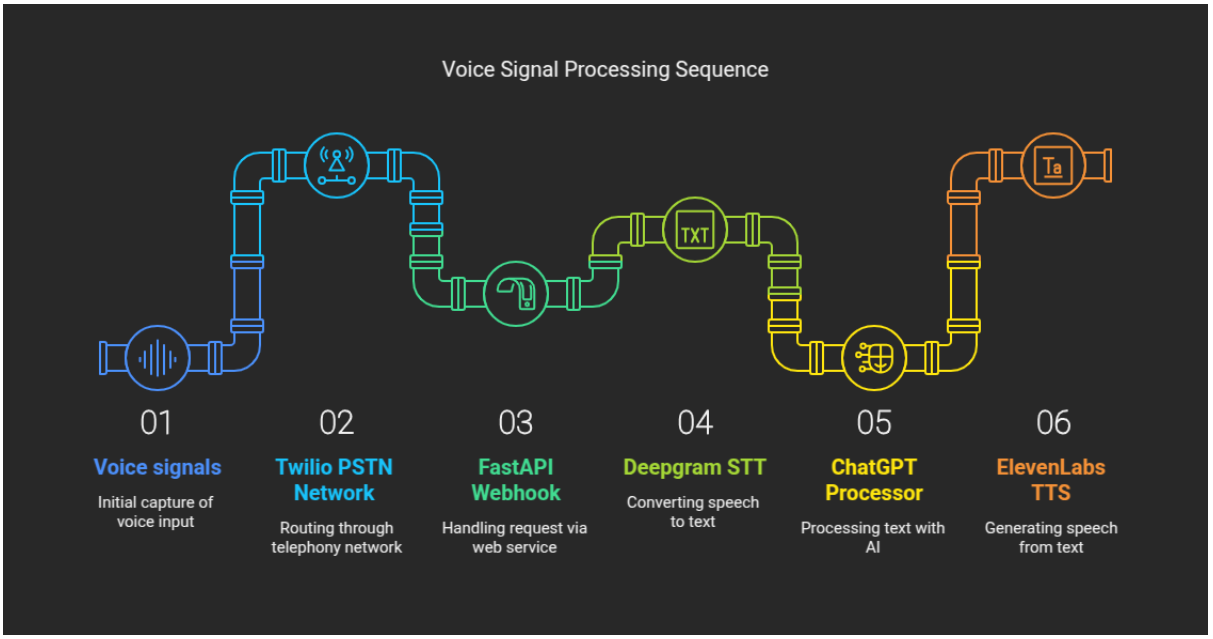
The telephony system comprises four principal modules that interact through defined APIs and event-driven workflows[1][2]:

1. **Telephony Server (FastAPI)**
   Acts as the central hub managing voice communication channels and API endpoints. Built on FastAPI's ASGI implementation, it handles Webhook integrations with Twilio's PSTN connectivity while maintaining conversation state through Redis[2][5]. The server utilizes Python's async/await paradigm to manage concurrent call sessions efficiently.
2. **AI Conversation Agent**
   Implements a dual-layer architecture combining:
   - **Natural Language Processing (ChatGPT)**
     Processes text inputs using OpenAI's language models to generate context-aware responses
   - **Speech Processing Pipeline**
     Integrates Deepgram for speech-to-text conversion and ElevenLabs for text-to-speech synthesis[3][5]
3. **Event Management System**

Implements an observer pattern through the EventsManager class, handling transcript completion events and persisting conversation data to structured JSON storage. The system employs type-safe event casting and atomic file operations to ensure data integrity[1].

4. **Deployment Infrastructure**
   Utilizes containerized Redis for configuration management and ngrok for secure tunneling of local development environments. The architecture supports both cloud-native deployments and hybrid on-premises configurations[4][5].

## Data Flow Diagram



# Telecommunication and Processing Flow

# API Integration Matrix

## Third-Party Services

| Service | Purpose | Authentication Method | Rate Limit | Documentation |
|---|---|---|---|---|
| Twilio | PSTN Connectivity | Account SID/Auth Token | 1 QPS/core | Twilio REST API[5] |
| OpenAI | Text Generation | API Key | 3.5k TPM | ChatGPT API[1] |
| Deepgram | Speech Recognition | API Key | 50 hrs/mo | Streaming API[3] |
| ElevenLabs | Voice Synthesis | API Key | 10k chars/mo | Text-to-Speech API[5] |
| Redis | State Management | None (local) | Memory-bound | Redis Commands[4] |

## Custom API Endpoints

```python
@app.post("/inbound_call")
async def handle_twilio_webhook(request: Request):
    """Entry point for Twilio voice webhooks"""
    return await telephony_server.handle_twilio_webhook(request)

@app.get("/call/{conversation_id}")
async def get_call_status(conversation_id: str):
    """Retrieve conversation state from Redis"""
    return config_manager.get_config(conversation_id)
```

The system implements JWT-free authentication through Twilio's request signature validation and environment variable segregation. All sensitive credentials are stored in `.env` following 12-factor app principles[5].

# Architecture

## Containerized Services

```
docker run -dp 6379:6379 -it redis/redis-stack:latest
```

The Redis container provides:

- In-memory key-value store for active conversation states
- LRU cache eviction policy for resource optimization
- Persistence snapshots through RDB/AOF configurations

### Port Mapping Schema

| Component | Port | Protocol | Exposure |
|---|---|---|---|
| FastAPI | 3000 | HTTP | ngrok Tunnel |
| Redis | 6379 | TCP | Localhost Only |
| ngrok | 4040 | HTTP/TCP | Public Internet |

The ngrok integration implements automatic tunnel provisioning with auth token validation[4]:

```
if not BASE_URL and ngrok_auth:
    ngrok.set_auth_token(ngrok_auth)
    BASE_URL = ngrok.connect(port).public_url
    logger.info(f"Tunnel established: {BASE_URL}")
```

# Installation Guide

### Prerequisite Setup

1. **Twilio Configuration**
    - Purchase phone number in Console
    - Enable "Voice" capability
    - Set Webhook URL to ngrok endpoint
2. **AI Service Accounts**

```
echo "ELEVEN_LABS_API_KEY=your_key" >> .env
echo "DEEPGRAM_API_KEY=your_key" >> .env
echo "OPENAI_API_KEY=your_key" >> .env
```

3. **Voice Model Selection**

Edit `ElevenLabsSynthesizerConfig`:

```
ElevenLabsSynthesizerConfig(
```

```
    voice_id="EXAVITQu4vr4xnSDxMaL",
    stability=0.71,
    similarity_boost=0.45,
    model_id="eleven_monolingual_v2"
)
```

### Runtime Management

Start services in ordered sequence:

```
# Terminal 1: Redis
docker run -dp 6379:6379 redis

# Terminal 2: FastAPI
poetry run uvicorn main:app --port 3000 --workers 2

# Terminal 3: Outbound Calls
python -m outbound_calls
```

## Monitoring and Diagnostics

### Log Analysis

```
configure_pretty_logging(
    level="DEBUG",
    enable_json=False,
    enable_cloud_logging=False
)
```

Log Levels:

- **DEBUG**: Full conversation transcripts
- **INFO**: Call lifecycle events
- **WARNING**: API rate limit alerts
- **ERROR**: STT/TTS failures

### Transcript Storage

The EventsManager implements atomic JSON writes:

```
def store_transcript_to_file(data: dict):
    with open("transcripts.json", "r+") as f:
```

```
        transcripts = json.load(f)
        transcripts[data["id"]] = data
        f.seek(0)
        json.dump(transcripts, f, indent=2)
        f.truncate()
```

Schema:

```
{
  "conversation_id": "uuid4",
  "user_id": 123,
  "transcript": "text",
  "timestamp": "ISO8601",
  "duration": 142.7
}
```

# Performance Optimization

## Audio Processing Parameters

```
ElevenLabsSynthesizerConfig(
    sampling_rate=24000,
    audio_encoding="mulaw",
    chunk_size=2048,
    stream_timeout=30
)

DeepgramTranscriberConfig(
    model="nova-2",
    language="en-US",
    endpointing="punctuation",
    interim_results=True
)
```

## Conversation Timeouts

```
ChatGPTAgentConfig(
    allowed_idle_time_seconds=30,
    num_check_human_present_times=3,
    interrupt_sensitivity="high",
    end_conversation_on_goodbye=True
)
```

# Security Considerations

### Twilio Request Validation

```
from fastapi.security import HTTPBearer
security = HTTPBearer(auto_error=False)

async def verify_twilio_signature(request: Request):
    validator = RequestValidator(os.environ['TWILIO_AUTH_TOKEN'])
    return validator.validate(
        str(request.url),
        await request.body(),
        request.headers.get('X-TWILIO-SIGNATURE')
    )
```

### Secret Management

- Environment variables stored in `.env`
- Git-ignored from version control
- Encrypted in production using Vault/SSM

### TLS Configuration

```
poetry add uvicorn[standard]
uvicorn main:app --ssl-keyfile key.pem --ssl-certfile cert.pem
```

# Scaling Strategies

## Horizontal Scaling

```
telephony_server = TelephonyServer(
    base_url=os.getenv("LOAD_BALANCER_URL"),
    config_manager=RedisClusterConfigManager(
        nodes=[{"host": "redis-node1", "port": 6379}]
    ),
    inbound_call_configs=[...],
    worker_count=os.cpu_count() * 2
)
```

## Performance Metrics

| Metric | Monitoring Tool | Alert Threshold |
|---|---|---|
| API Latency | Prometheus | >2000ms |
| Error Rate | Grafana | >5% |
| Active Calls | CloudWatch | >80% Capacity |
| CPU Usage | Datadog | >75% |

# Conclusion

This architecture demonstrates an enterprise-grade telephony solution combining modern AI services with robust infrastructure components. The system's modular design allows for:

1. **Component Swapping**
    - Replace Deepgram with Azure Speech Services
    - Substitute ElevenLabs with Amazon Polly
    - Migrate Redis to managed cloud service
2. **Feature Extensions**
    - Add SMS handling via Twilio Programmable Messaging
    - Implement sentiment analysis on conversation transcripts
    - Integrate CRM systems through REST API adapters
3. **Compliance Enhancements**

- GDPR-compliant data anonymization
- HIPAA-ready encryption protocols
- PCI DSS payment processing integration

Future development should focus on implementing adaptive voice models and multi-lingual support while maintaining the system's low-latency characteristics[1][3]. Regular security audits and performance benchmarking are recommended for production deployments[5].