# When Content Speaks Volumes: Podcastfy — An Open Source Python Package Bridging Multimodal Data and Conversational Audio with GenAI

Tharsis T. P. Souza [1,2]

1 Columbia University in the City of New York **2** Instituto Federal de Educacao, Ciencia e Tecnologia do Sul de Minas (IFSULDEMINAS)

Abstract

Podcastfy is an open-source Python framework that programmatically transforms multisourced, multimodal content into multilingual, natural-sounding audio conversations using generative AI. By converting various types of digital content - including images, websites, YouTube videos, and PDFs - into conversational audio formats, Podcastfy enhances accessibility, engagement, and usability for a wide range of users. As an open-source project, Podcastfy benefits from continuous community-driven improvements, enhancing its adaptability to evolving user requirements and accessibility standards.

Statement of Need

The rapid expansion of digital content across various formats has intensified the need for tools capable of converting diverse information into accessible and digestible forms (Chen & Wu, 2023; Johnson & Smith, 2023; McCune & Brown, 2023). Existing solutions often fall short due to their proprietary nature, limited multimodal support, or inadequate accessibility features (Gupta & Lee, 2023; Marcus & Zhang, 2019; Peterson & Allen, 2023).

Podcastfy addresses this gap with an open-source solution that supports multimodal input processing and generates natural-sounding, summarized conversational content. Leveraging advances in large language models (LLMs) and text-to-speech (TTS) synthesis, Podcastfy aims to benefit a diverse group of users — including content creators, educators, researchers, and accessibility advocates — by providing a customizable solution that transforms digital content into multilingual textual and auditory formats, enhancing accessibility and engagement.

Features

- Generate conversational content from multiple sources and formats (images, websites, YouTube, and PDFs).
- Customize transcript and audio generation (e.g., style, language, structure, length).
- Create podcasts from pre-existing or edited transcripts.
- Leverage cloud-based and local LLMs for transcript generation (increased privacy and control).
- Integrate with advanced text-to-speech models (OpenAI, ElevenLabs, and Microsoft Edge).
- Provide multi-language support for global content creation and enhanced accessibility.
- Integrate seamlessly with CLI and Python packages for automated workflows.

See audio samples.

Use Cases

Podcastfy is designed to serve a wide range of applications, including:

- **Content Creators** can use `Podcastfy` to convert blog posts, articles, or multimedia content into podcast-style audio, enabling them to reach broader audiences. By transforming content into an audio format, creators can cater to users who prefer listening over reading.

- **Educators** can transform lecture notes, presentations, and visual materials into audio conversations, making educational content more accessible to students with different learning preferences. This is particularly beneficial for students with visual impairments or those who have difficulty processing written information.

- **Researchers** can convert research papers, visual data, and technical content into conversational audio. This makes it easier for a wider audience, including those with disabilities, to consume and understand complex scientific information. Researchers can also create audio summaries of their work to enhance accessibility.

- **Accessibility Advocates** can use `Podcastfy` to promote digital accessibility by providing a tool that converts multimodal content into auditory formats. This helps individuals with visual impairments, dyslexia, or other disabilities that make it challenging to consume written or visual content.

Implementation and Architecture

Podcastfy implements a modular architecture designed for flexibility and extensibility through five main components, as shown in Figure 1.

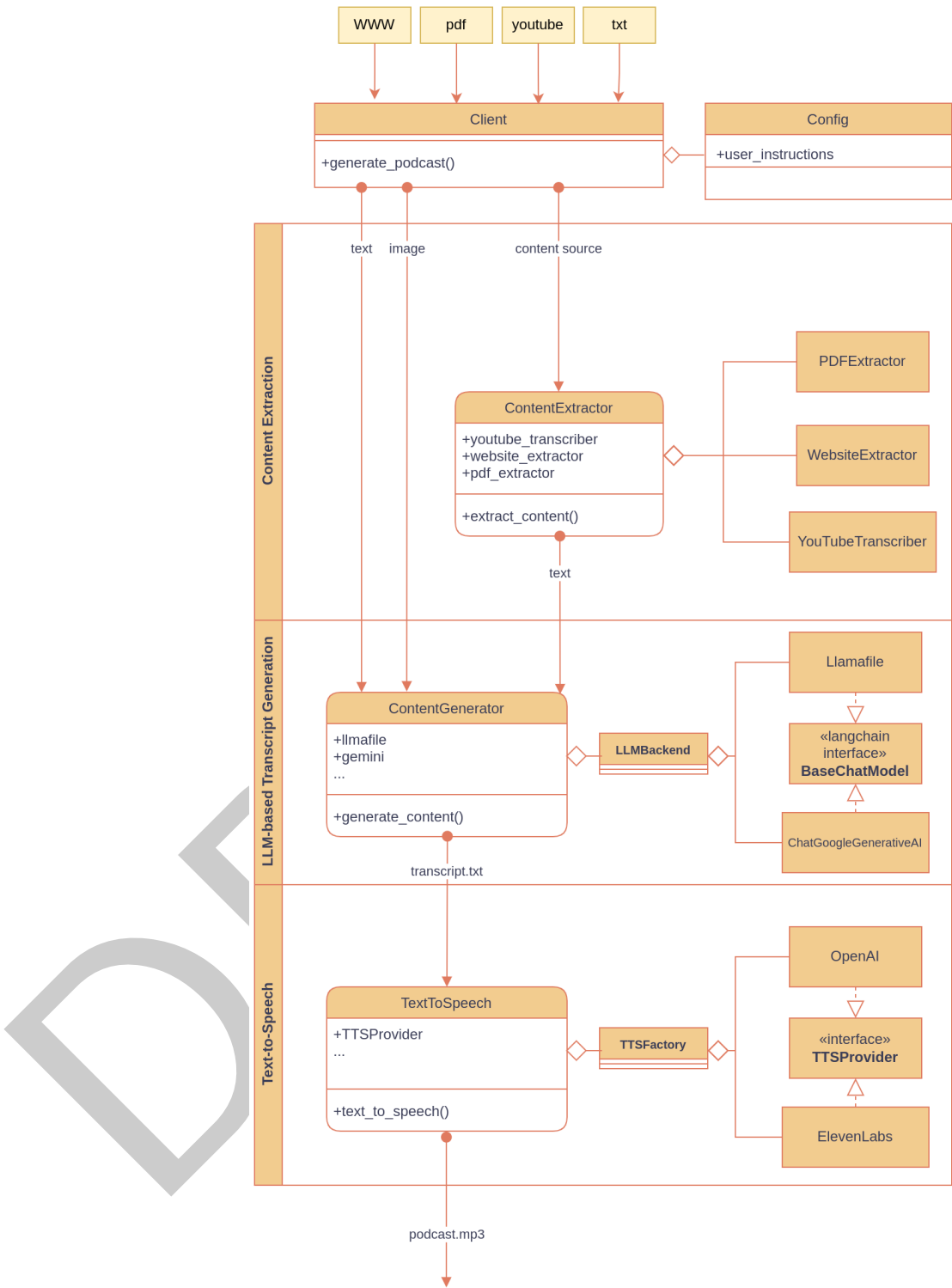**Figure 1:** Podcastfy's architecture and workflow diagram showing the main components and their interactions.

1. **Client Interface**
   - Provides both CLI (Command-Line Interface) and API interfaces.
   - Coordinates the workflow between processing layers.

- - Implements a unified interface for podcast generation through the `generate_podcast()` method.
  2. **Configuration Management**
     - Offers extensive customization options through a dedicated module.
     - Manages system settings and user preferences, such as podcast name, language, style, and structure.
     - Controls the behavior of all processing layers.
  3. **Content Extraction Layer**
     - Extracts content from various sources, including websites, PDFs, and YouTube videos.
     - The `ContentExtractor` class coordinates three specialized extractors:
       - `PDFExtractor`: Handles PDF document processing.
       - `WebsiteExtractor`: Manages website content extraction.
       - `YouTubeTranscriber`: Processes YouTube video content.
     - Serves as the entry point for all input types, providing standardized text output to the transcript generator.
  4. **LLM-based Transcript Generation Layer**
     - Uses large language models to generate natural-sounding conversations from extracted content.
     - The `ContentGenerator` class manages conversation generation using different LLM backends:
       - Integrates with LangChain to implement prompt management and common LLM access through the `BaseChatModel` interface.
       - Supports both local (`LlamaFile`) and cloud-based models.
       - Uses `ChatGoogleGenerativeAI` for cloud-based LLM services.
     - Allows customization of conversation style, roles, and dialogue structure.
     - Outputs structured conversations in text format.
  5. **Text-to-Speech (TTS) Layer**
     - Converts input transcripts into audio using various TTS models.
     - The `TextToSpeech` class implements a factory pattern:
       - The `TTSFactory` creates appropriate providers based on configuration.
       - Supports multiple backends (OpenAI, ElevenLabs, and Microsoft Edge) through the `TTSProvider` interface.
     - Produces the final podcast audio output.

The modular architecture enables independent development and maintenance of each component. This pipeline design ensures a clean separation of concerns while maintaining seamless data transformation between stages. This modular approach also facilitates easy updates and extensions to individual components without affecting the rest of the system.

The framework is offered as a Python package, with a command-line interface as well as a REST API, making it accessible to users with different technical backgrounds and requirements.

Quick Start

Prerequisites

- Python 3.11 or higher
- `$ pip install ffmpeg` (for audio processing)

Setup

1. Install from PyPI `$ pip install podcastfy`

2. Set up [API keys](#)

Python

```python
from podcastfy.client import generate_podcast

audio_file = generate_podcast(urls=["<url1>", "<url2>"])
```

CLI

```
python -m podcastfy.client --url <url1> --url <url2>
```

Customization Examples

Podcastfy offers various customization options that make it versatile for different types of content transformation. To accomplish that, we leverage LangChain's (LangChain, 2024) prompt management capabilities to dynamically construct prompts for the LLM, adjusting conversation characteristics such as style, roles, and dialogue structure. Below are some examples that demonstrate its capabilities.

Academic Debate

The following Python code demonstrates how to configure Podcastfy for an academic debate:

```python
from podcastfy import generate_podcast

debate_config = {
    "conversation_style": ["formal", "debate"],
    "roles_person1": "main presenter",
    "roles_person2": "opposing viewpoint",
    "dialogue_structure": ["Introduction", "Argument Presentation", "Counterarguments",
}

generate_podcast(
    urls=["PATH/TO/academic-article.pdf"],
    conversation_config=debate_config
)
```

In this example, the roles are set to "main presenter" and "opposing viewpoint" to simulate an academic debate between two speakers on a chosen topic. This approach is especially useful for educational content that aims to present multiple perspectives on a topic. The output is structured with clear sections such as introduction, argument presentation, counterarguments, and conclusion, allowing listeners to follow complex ideas easily.

Storytelling Adventure

The following Python code demonstrates how to generate a storytelling podcast:

```python
from podcastfy import generate_podcast

story_config = {
    "conversation_style": ["adventurous", "narrative"],
    "creativity": 1.0,
    "roles_person1": "narrator",
    "roles_person2": "character",
    "dialogue_structure": ["Introduction", "Adventure Begins", "Challenges", "Resolution
}
```

```
generate_podcast(
    urls=["SAMPLE/WWW.URL.COM"],
    conversation_config=story_config
)
```

128 In this example, Podcastfy creates an engaging story by assigning roles like "narrator" and
129 "character" and adjusting the creativity parameter for richer descriptions. Using this con-
130 figuration, Podcastfy can generate engaging narrative content. By adjusting the creativity
131 parameter, Podcastfy can create a story involving multiple characters, unexpected plot twists,
132 and rich descriptions.

133 **Additional Examples**

134 **Daily News Briefing**

```
news_config = {
    "word_count": 1500,
    "conversation_style": ["concise", "informative"],
    "podcast_name": "Morning Briefing",
    "dialogue_structure": [
        "Headlines",
        "Key Stories",
        "Market Update",
        "Weather"
    ],
    "roles_person1": "news anchor",
    "roles_person2": "field reporter",
    "creativity": 0.3
}

generate_podcast(
    urls=[
        "https://news-source.com/headlines",
        "https://market-updates.com/today"
    ],
    conversation_config=news_config
)
```

135 **Language Learning Content**

```
language_config = {
    "output_language": "Spanish",
    "word_count": 1000,
    "conversation_style": ["educational", "casual"],
    "engagement_techniques": [
        "vocabulary explanations",
        "cultural context",
        "pronunciation tips"
    ],
    "roles_person1": "language teacher",
    "roles_person2": "curious student",
    "creativity": 0.6
}

generate_podcast(
    urls=["https://spanish-content.com/article"],
```

```
        conversation_config=language_config
)
```

Technical Tutorial

```python
tutorial_config = {
    "word_count": 2500,
    "conversation_style": ["instructional", "step-by-step"],
    "roles_person1": "expert developer",
    "roles_person2": "learning developer",
    "dialogue_structure": [
        "Concept Introduction",
        "Technical Background",
        "Implementation Steps",
        "Common Pitfalls",
        "Best Practices"
    ],
    "engagement_techniques": [
        "code examples",
        "real-world applications",
        "troubleshooting tips"
    ],
    "creativity": 0.4
}

generate_podcast(
    urls=["https://tech-blog.com/tutorial"],
    conversation_config=tutorial_config
)
```

Working with Podcastfy Modules

Podcastfy's components are designed to work independently, allowing flexibility in updating or extending each module. The data flows from the `ContentExtractor` module to `ContentGenerator` and finally to the `TexttoSpeech` converter, ensuring a seamless transformation of multimodal content into audio. In this section, we provide some examples of how to use each module.

Content Extraction

Podcastfy's `content_extractor.py` module allows users to extract content from a given URL, which can be processed further to generate a podcast. Below is an example of how to use the content extraction component:

```python
from podcastfy.content_extractor import ContentExtractor

# Initialize the content extractor
extractor = ContentExtractor()

# Extract content from a URL
url = "https://example.com/article"
extracted_content = extractor.extract_content(url)

print("Extracted Content:")
print(extracted_content)
```

147 This example demonstrates how to extract text from a given URL. The extracted content is
148 then passed to the next stages of processing.

Content Generation

150 The `content_generator.py` module is responsible for generating conversational content based
151 on textual input. Below is an example of how to use the content generation component:

```python
from podcastfy.content_generator import ContentGenerator

# Initialize the content generator
generator = ContentGenerator(api_key="<GEMINI_API_KEY>")

# Generate conversational content
input_text = "This is a sample input text about artificial intelligence."
generated_conversation = generator.generate_conversation(input_text)

print("Generated Conversation:")
print(generated_conversation)
```

152 Users can opt to run a cloud-based LLM (Gemini) or run a local (potentially Open Source)
153 LLM model (see local llm configuration).

Text-to-Speech Conversion

155 The `text_to_speech.py` module allows the generated transcript to be converted into audio.
156 Below is an example of how to use the text-to-speech component:

```python
from podcastfy.text_to_speech import TextToSpeech

# Initialize the text-to-speech converter
tts = TextToSpeech(model='elevenlabs', api_key="<ELEVENLABS_API_KEY>")

# Convert the generated conversation to speech
input_text = "<Person1>This is a sample conversation generated by Podcastfy.</Person1><P
output_audio_file = "output_podcast.mp3"
tts.convert_to_speech(input_text, output_audio_file)

print(f"Audio saved to {output_audio_file}")
```

157 This example demonstrates how to use the `TextToSpeech` class to convert generated text into
158 an audio file. Users can specify different models for TTS, such as `elevenlabs`, `openai`, or
159 `edge` (free to use).

Limitations

161 Podcastfy has several limitations, including:

162 - **Content Accuracy and Quality**
163   - The accuracy of generated conversations depends heavily on the capabilities of the
164   underlying LLMs.
165   - Complex technical or domain-specific content may not always be accurately inter-
166   preted or summarized.
167   - The framework cannot guarantee the factual correctness of generated content,
168   requiring human verification for critical applications.
169 - **Language Support Constraints**

170   – While multilingual support is available, performance may vary significantly across
171   different languages.
172   – Less common languages may have limited TTS voice options and lower-quality
173   speech synthesis.
174   – Nuanced cultural contexts and idioms may not translate effectively across languages.

175   ■ **Technical Dependencies**
176   – Reliance on third-party APIs (OpenAI, ElevenLabs, Google) introduces potential
177   service availability risks.
178   – Local LLM options, while providing independence, require significant computational
179   resources.
180   – Network connectivity is required for cloud-based services, limiting offline usage.

181   ■ **Content Extraction Challenges**
182   – Complex webpage layouts or dynamic content may not be accurately extracted.
183   – PDF extraction quality depends on document formatting and structure.
184   – YouTube video processing depends on the availability of transcripts.

185   ■ **Accessibility Considerations**
186   – Generated audio may not fully meet all accessibility standards.
187   – Limited support for real-time content processing.
188   – May require additional processing for users with specific accessibility needs.

189 These limitations highlight areas for future development and improvement of the framework.
190 Users should carefully consider these constraints when implementing `Podcastfy` for their
191 specific use cases and requirements.

Conclusion

193 `Podcastfy` contributes to multimodal content accessibility by enabling the programmatic
194 transformation of digital content into conversational audio. The framework addresses accessi-
195 bility needs through automated content summarization and natural-sounding speech synthesis.
196 Its modular design and configurable options allow for flexible content processing and audio
197 generation workflows that can be adapted for different use cases and requirements.

198 As an open-source project, `Podcastfy` benefits from continuous community-driven improve-
199 ments and adaptations, helping support its long-term value and relevance in meeting evolving
200 user requirements and accessibility standards.

201 We invite contributions from the community to further enhance the capabilities of `Podcastfy`.
202 Whether it's by adding support for new input modalities, improving the quality of conversation
203 generation, or optimizing the TTS synthesis, we welcome collaboration to make `Podcastfy`
204 more powerful and versatile.

Acknowledgements

References

211 Chen, R., & Wu, Y. (2023). Digital accessibility tools for multimodal content processing: A
212   systematic review. *Digital Transformation Review*, *5*(3), 91–109.

213 Gupta, S., & Lee, A. (2023). Advances in adaptive user interfaces for enhanced accessibility.
214   *Journal of Accessibility and User Experience*, *14*(3), 203–215.

Johnson, L., & Smith, K. (2023). Adaptive user interfaces for accessibility across digital content modalities. *Journal of Multimodal Accessibility*, *17*(2), 43–58. https://link.springer.com/article/10.1007/s12193-023-00427-4

LangChain. (2024). *LangChain: Building applications with LLMs through composability*. https://www.langchain.com/

Marcus, A., & Zhang, T. (2019). Design for multimodal human-computer interaction. In *Lecture notes in computer science* (Vol. 1157, pp. 160–176). https://link.springer.com/chapter/10.1007/978-3-319-52162-6_18

McCune, B., & Brown, L. (2023). Accessibility of digital information: Standards, frameworks, and tools related to information literacy and information technology. *Journal of Information Literacy and Accessibility*, *9*(4), 112–129.

Peterson, L., & Allen, J. (2023). Web accessibility and multimodal digital engagement. *Technology Accessibility Journal*, *12*(1), 23–34.