# Yankovinator: Song Parody Generator

Technical Documentation

Shyamal Suhana Chandra

Copyright © 2025

## Contents

**1 Introduction** **3**

**2 Architecture** **3**
2.1 Core Components . . . 3
2.2 Technology Stack . . . 3

**3 Usage** **3**
3.1 Command Line Interface . . . 3
3.2 Programmatic Usage . . . 3

**4 API Reference** **4**
4.1 SyllableCounter . . . 4
4.1.1 countSyllables(in:) . . . 4
4.1.2 countSyllablesInLine(_:) . . . 4
4.1.3 analyzeSongStructure(_:) . . . 4
4.2 ParodyGenerator . . . 4
4.2.1 generateParody(originalLyrics:keywords:progressCallback:refinementPasses:verbose:) . . . 4
4.3 Yankovinator . . . 5
4.3.1 generateParody(originalLyrics:keywords:ollamaURL:ollamaModel:) . . . 5

**5 Algorithm** **5**
5.1 Syllable Counting . . . 5
5.2 Rhyme Detection . . . 5
5.3 Parody Generation . . . 5
5.4 Semantic Coherence . . . 6
5.5 Text Processing and Grammar . . . 6
5.6 Capitalization and Punctuation Matching . . . 6

**6 Documentation** **7**

**7 Testing** **7**

**8 Installation** **7**
8.1 Requirements . . . 7
8.2 Homebrew Installation (Recommended) . . . 7
8.3 Build from Source . . . 8

**9 Configuration** **8**
9.1 Ollama Setup . . . 8

**10 Examples** **8**
10.1 Example Input . . . 8
10.2 Example Output . . . 8

**11 Troubleshooting** **8**
11.1 Common Issues . . . . . 8

**12 License** **8**

**13 References** **9**

# 1 Introduction

Yankovinator is a Swift-based application that converts songs into parodies using Apple's NaturalLanguage framework and Ollama for intelligent text generation. The system maintains the original song's syllable structure while generating new lyrics that follow theme-based keyword constraints.

# 2 Architecture

## 2.1 Core Components

Yankovinator consists of four main components:

1. **SyllableCounter**: Analyzes syllable structure using NaturalLanguage framework
2. **OllamaClient**: Interfaces with Ollama API for parody generation
3. **ParodyGenerator**: Orchestrates the parody creation process
4. **Yankovinator**: Main library interface

## 2.2 Technology Stack

- **Swift 6.2+**: Primary programming language
- **NaturalLanguage**: Apple framework for linguistic analysis
- **Ollama**: Local LLM for text generation
- **AsyncHTTPClient**: Asynchronous HTTP client for API communication
- **ArgumentParser**: CLI argument parsing

# 3 Usage

## 3.1 Command Line Interface

The CLI tool provides a simple interface for generating parodies:

```
yankovinator <lyrics-file> [--keywords <keywords-file>]
            [--ollama-url <url>] [--model <model-name>]
            [--output <output-file>] [--analyze] [--verbose]
```

## 3.2 Programmatic Usage

```swift
import Yankovinator

let lyrics = [
    "Twinkle twinkle little star",
    "How I wonder what you are"
]

let keywords = [
    "space": "the physical universe beyond Earth",
    "stars": "luminous celestial bodies"
]

let parody = try await Yankovinator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords
)
```

# 4 API Reference

## 4.1 SyllableCounter

### 4.1.1 countSyllables(in:)

Counts syllables in a single word.

**Parameters:**

- `word: String` - The word to analyze

**Returns:** `Int` - Number of syllables

### 4.1.2 countSyllablesInLine(_:)

Counts total syllables in a line of text.

**Parameters:**

- `line: String` - The line of text

**Returns:** `Int` - Total syllable count

### 4.1.3 analyzeSongStructure(_:)

Analyzes syllable structure of a complete song.

**Parameters:**

- `lyrics: [String]` - Array of lyric lines

**Returns:** `[Int]` - Array of syllable counts per line

## 4.2 ParodyGenerator

### 4.2.1 generateParody(originalLyrics:keywords:progressCallback:refinementPasses:verbose:)

Generates a parody matching the original song's structure with semantic coherence.

**Parameters:**

- `originalLyrics: [String]` - Original song lines
- `keywords: [String: String]` - Theme keywords and definitions
- `progressCallback: ((Int, Int) -> Void)?` - Optional progress callback
- `refinementPasses: Int` - Number of refinement passes (default: 2)
- `verbose: Bool` - Whether to print verbose messages

**Returns:** `[String]` - Generated parody lines with semantic coherence

**Refinement Process:**

1. Word-by-word syllable matching refinement
2. Semantic coherence refinement (ensures lines work together thematically)
3. Capitalization and punctuation matching to match original style exactly

### 4.3 Yankovinator

#### 4.3.1 generateParody(originalLyrics:keywords:ollamaURL:ollamaModel:)

Main entry point for parody generation.

**Parameters:**

- `originalLyrics: [String]` - Original song lines
- `keywords: [String: String]` - Theme keywords and definitions
- `ollamaURL: String` - Ollama API base URL (default: http://localhost:11434)
- `ollamaModel: String` - Ollama model name (default: llama3.2)

**Returns:** `[String]` - Generated parody lines

## 5 Algorithm

### 5.1 Syllable Counting

The syllable counting algorithm uses a combination of:

1. NaturalLanguage tokenization for word separation
2. Vowel counting heuristics
3. Special case handling for silent 'e' and 'le' endings
4. Word-by-word syllable analysis for precise matching

### 5.2 Rhyme Detection

Yankovinator automatically detects rhyming schemes from original lyrics:

- Extracts last words from each line using NaturalLanguage tokenization
- Compares phonetic patterns using vowel sequences and consonant patterns
- Groups lines by rhyming patterns (A, B, C, etc.)
- Enforces rhyming constraints during generation
- Supports common rhyme schemes: ABAB, AABB, AAAA, etc.

### 5.3 Parody Generation

The parody generation process:

1. Analyze original song's syllable structure
2. Detect rhyming scheme from original lyrics
3. For each line:
   (a) Extract syllable count requirement (total and word-by-word)
   (b) Determine rhyming constraints based on detected scheme
   (c) Build prompt with theme keywords and semantic context
   (d) Request generation from Ollama with previous lines for coherence
   (e) Validate and clean response
   (f) Refine word-by-word syllable matching
   (g) Refine semantic coherence with previous lines
   (h) Apply capitalization and punctuation to match original style exactly
4. Return complete parody

### 5.4 Semantic Coherence

Yankovinator employs advanced semantic coherence techniques to ensure generated lyrics work together meaningfully:

- **Context Awareness**: Each line generation considers up to 8 previous lines for narrative continuity
- **Theme Advancement**: Lines are generated to actively advance the chosen theme, not just mention keywords
- **Semantic Refinement**: A dedicated refinement pass ensures each line semantically connects with previous lines
- **Narrative Flow**: The system maintains logical progression and consistent imagery throughout the parody
- **Theme Integration**: Keywords are woven naturally into the semantic content of each line

The semantic coherence system uses Ollama's language understanding to:

- Build upon established narrative, theme, and emotional arcs
- Use consistent imagery, metaphors, and thematic elements
- Ensure each line contributes meaningfully to the overall story
- Maintain logical flow and progression from previous lines

### 5.5 Text Processing and Grammar

Yankovinator includes sophisticated text processing to ensure grammatically correct output:

- **Contraction Preservation**: Apostrophes in contractions (e.g., "don't", "can't", "it's", "won't") are preserved during text cleaning, ensuring natural and grammatically correct poetry
- **Quote Handling**: Only wrapping quotes are removed from generated text, while internal punctuation and apostrophes are maintained
- **Syllable Counting with Contractions**: The syllable counter properly handles contractions by tokenizing them as single words, then removing punctuation for accurate syllable counting
- **Prompt Engineering**: Generation prompts explicitly encourage the use of proper contractions when appropriate for natural speech patterns

### 5.6 Capitalization and Punctuation Matching

Yankovinator ensures that generated parodies match the exact capitalization and punctuation style of the original song:

- **Capitalization Pattern Extraction**: The system analyzes which words in the original line are capitalized and which are lowercase
- **Punctuation Preservation**: All punctuation marks (commas, periods, apostrophes, quotes, etc.) are extracted and preserved in their exact positions
- **Pattern Application**: After generation and refinement, the capitalization and punctuation patterns from the original line are automatically applied to the generated line
- **Word-by-Word Matching**: Each word in the generated line receives the same capitalization treatment as the corresponding word in the original line
- **Spacing Preservation**: The exact spacing and punctuation placement between words is maintained

The capitalization and punctuation matching process:

1. Extract capitalization pattern from original line (which words are uppercase/lowercase)
2. Extract punctuation pattern (all punctuation marks and their positions)
3. Generate parody line with semantic coherence and syllable matching
4. Apply capitalization pattern word-by-word to generated line
5. Apply punctuation pattern to match original exactly
6. Validate final output matches original style

# 6 Documentation

Yankovinator includes comprehensive documentation:

- **Technical Paper** (this document): Architecture, algorithms, and implementation details
- **Presentation**: Slide deck with overview, features, and usage examples
- **Reference Manual**: Complete API documentation with function signatures, parameters, return types, and usage examples

All documentation is available on GitHub Pages and can be generated locally using LaTeX.

# 7 Testing

The project includes comprehensive tests using XCTest:

- Unit tests for syllable counting
- Integration tests for parody generation
- Keyword extraction tests

Run tests with:

```
swift test
```

# 8 Installation

## 8.1 Requirements

- Swift 6.3 or later
- macOS 13+ or iOS 16+
- Ollama installed and running
- Model: llama3.2:3b (install with `ollama pull llama3.2:3b`)

## 8.2 Homebrew Installation (Recommended)

The easiest way to install Yankovinator is using Homebrew:

```
# Add the tap
brew tap shyamalschandra/yankovinator

# Install Yankovinator
brew install yankovinator

# Verify installation
yankovinator --help
keyword-generator --help
```

The Homebrew formula automatically handles all dependencies and builds the package in release mode.

### 8.3 Build from Source

For development or if you prefer building from source:

```
# Clone repository
git clone https://github.com/shyamalschandra/Yankovinator.git
cd yankovinator

# Build
swift build

# Run CLI
swift run yankovinator --help
```

## 9 Configuration

### 9.1 Ollama Setup

Ensure Ollama is running and accessible:

```
# Start Ollama
ollama serve

# Pull required model
ollama pull llama3.2
```

## 10 Examples

### 10.1 Example Input

**lyrics.txt:**

```
Twinkle twinkle little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
```

**keywords.txt:**

```
science: the study of natural phenomena
space: the physical universe beyond Earth
exploration: the action of traveling to discover
```

### 10.2 Example Output

The generated parody maintains syllable structure while incorporating the theme keywords.

## 11 Troubleshooting

### 11.1 Common Issues

- **Ollama not found**: Ensure Ollama is running at the specified URL
- **Model not available**: Pull the required model using `ollama pull`
- **Syllable count mismatch**: The algorithm uses heuristics and may not be 100% accurate

## 12 License

Copyright © 2025, Shyamal Suhana Chandra

## 13 References


- Apple NaturalLanguage Framework: `https://developer.apple.com/documentation/NaturalLanguage`
- Ollama Documentation: `https://ollama.ai/docs`
- Swift Package Manager: `https://swift.org/package-manager/`