# Babylon.cpp Manual

v1.0.0

March 9, 2026

# Contents

# 1 Introduction

Babylon.cpp is a free and open-source C and C++ library for grapheme-to-phoneme (G2P) conversion and neural text-to-speech (TTS) synthesis. All inference runs locally using ONNX Runtime — no internet connection is required and no text or audio data leaves the host machine.

The library exposes three layers of interface:

- A **C API** suitable for use from any language with a C foreign-function interface.
- A **C++ API** providing higher-level session classes.
- A **command-line tool** (`babylon`) for phonemization, speech synthesis, and serving a REST API.

## 1.1 Supported Platforms

| Platform | Architecture | Library |
|---|---|---|
| Linux | x86_64 | `libbabylon.so` |
| macOS | Universal (x86_64 + arm64) | `libbabylon.dylib` |
| Windows | x86_64 | `babylon.dll` |
| Android | arm64-v8a, x86_64 | `libbabylon.so` |

# 2 How It Works

## 2.1 Phonemization

The G2P pipeline converts raw text to IPA phonemes through three stages:

1. **Text normalisation** — Numbers are expanded to words, ordinals are resolved, abbreviations are handled, and punctuation is classified so the phonemizer receives clean prose.

2. **Dictionary lookup** — Each word is looked up in a bundled pronunciation dictionary (~130 000 English entries). If found, the dictionary phonemes are used directly, avoiding a model call.

3. **Neural G2P** — Words not found in the dictionary are phonemized by Open Phonemizer, an ONNX model that predicts IPA output character-by-character using CTC decoding.

## 2.2 TTS Engines

Babylon.cpp supports two neural TTS backends.

### 2.2.1 Kokoro

Kokoro is the recommended engine. It produces high-quality, multi-voice speech at 24 kHz mono. Each voice is represented by a 256-dimensional style embedding stored in a `.bin` file, indexed by the token count of the input sequence.

The Kokoro synthesis pipeline:

1. Phonemize the input text to an IPA string.

2. Encode the IPA string to Kokoro token IDs using a built-in 178-entry vocab.

3. Load the voice style embedding for the current token count from the `.bin` file.

4. Run the Kokoro ONNX model with `input_ids`, `style`, and `speed` inputs to produce a PCM waveform.

5. Write the waveform as a 24 kHz WAV file.

#### 2.2.2 VITS

VITS is an end-to-end neural TTS model. Piper-compatible VITS models are supported. The sample rate is determined by metadata embedded in the model file.

## 3 Building from Source

### 3.1 Prerequisites

- CMake 3.18 or later
- A C++17 compiler (GCC, Clang, or MSVC)
- Git (for submodule checkout)
- **macOS only**: Xcode Command Line Tools
- **Windows only**: Visual Studio 2019 or later with the C++ workload

### 3.2 Cloning

```
git clone --recursive \
  https://github.com/Mobile-Artificial-Intelligence/babylon.cpp.git
cd babylon.cpp
```

### 3.3 Build Targets

| Target | Description |
|---|---|
| `make lib` | Build `libbabylon` only |
| `make cli` | Build the library, CLI binary, and copy runtime files to `bin/` |
| `make debug` | CLI build in Debug mode |
| `make android` | Cross-compile the library for Android (requires NDK) |

All build output is placed in `bin/`. The `cli` target also copies `data/config.json`, `data/index.html`, `data/dictionary.json`, and the `models/` directory into `bin/`.

### 3.4 Android

Android builds require the Android NDK (r27 or later). Set `ANDROID_NDK_HOME` before running `make android`.

## 4 Models

The library requires external model files that are not bundled in the repository. Place them in the `models/` directory, or configure their paths in `data/config.json`.

| File | Description | Config Key |
|---|---|---|
| `open-phonemizer.onnx` | Open Phonemizer G2P model | `phonemizer_model` |
| `dictionary.json` | Pronunciation dictionary | `dictionary` |
| `kokoro-quantized.onnx` | Kokoro TTS model | `kokoro_model` |
| `voices/*.bin` | Kokoro voice style files | `kokoro_voices` |
| `curie.onnx` | VITS TTS model | `vits_model` |

### 4.1 Kokoro Voices

Each voice is a `.bin` file of float32 values. The filename without extension is used as the voice name. The default voice is `en-US-heart`.

Voice names follow the convention `<lang>-<name>`:

| Language | Example Voice Names |
|---|---|
| English (US) | `en-US-heart`, `en-US-bella`, `en-US-nova`, `en-US-adam`, ... |
| English (GB) | `en-GB-alice`, `en-GB-emma`, `en-GB-daniel`, ... |
| German | `de-DE-dora`, `de-DE-alex` |
| French | `fr-FR-siwis` |
| Japanese | `ja-JP-alpha-f`, `ja-JP-kumo`, ... |
| Chinese (Simplified) | `zh-CN-xiaobei`, `zh-CN-yunxi`, ... |

## 5 CLI

The `babylon` binary provides three subcommands: `phonemize`, `tts`, and `serve`.

### 5.1 Global Options

The following options apply to all subcommands and are processed before dispatch:

| Option | Argument | Description |
|---|---|---|
| `-config` | `<path>` | Load a JSON config file |
| `-phonemizer-model` | `<path>` | Phonemizer ONNX model |
| `-dictionary` | `<path>` | Pronunciation dictionary JSON |
| `-kokoro-model` | `<path>` | Kokoro ONNX model |
| `-kokoro-voice` | `<name>` | Default Kokoro voice name |
| `-kokoro-voices` | `<dir>` | Directory of voice `.bin` files |
| `-vits-model` | `<path>` | VITS ONNX model |
| `-h`, `-help` | | Show help |

On startup, `babylon` automatically looks for a `config.json` in the same directory as the executable and loads it silently. A `-config` flag overrides this, and individual flags override specific keys.

## 5.2 Config File Format

```
{
  "phonemizer_model": "models/open-phonemizer.onnx",
  "dictionary":       "models/dictionary.json",
  "kokoro_model":     "models/kokoro-quantized.onnx",
  "kokoro_voice":     "en-US-heart",
  "kokoro_voices":    "models/voices",
  "vits_model":       "models/curie.onnx",
  "host":             "127.0.0.1",
  "port":             8775
}
```

## 5.3 phonemize

Convert text to IPA phonemes.

```
babylon phonemize "Hello␣world"
babylon phonemize --tokens "Hello␣world"
```

| Option | Description |
|---|---|
| -tokens | Print Kokoro token IDs instead of the IPA string |
| -h, -help | Show help |

## 5.4 tts

Synthesise speech and write a WAV file.

```
babylon tts "Hello␣world"
babylon tts "Hello␣world" -o hello.wav
babylon tts --voice en-US-nova --speed 1.2 "Hello␣world"
babylon tts --vits "Hello␣world" -o hello.wav
```

| Option | Argument | Description |
|---|---|---|
| -kokoro | | Use the Kokoro engine (default) |
| -vits | | Use the VITS engine |
| -engine | <name> | Select `kokoro` or `vits` explicitly |
| -v, -voice, -kokoro-voice | <name> | Kokoro voice name |
| -speed | <float> | Speech speed multiplier (default: `1.0`) |
| -o | <path> | Output WAV file (default: `output.wav`) |
| -h, -help | | Show help |

Voice names are filenames in the `kokoro_voices` directory without the `.bin` extension. For example, `-voice en-US-heart` maps to `models/voices/en-US-heart.bin`.

## 5.5 serve

Start a local REST API server with the web frontend.

```
babylon serve
babylon serve --host 0.0.0.0 --port 9000
```

| Option | Argument | Description |
|---|---|---|
| -host | <addr> | Bind address (default: 127.0.0.1) |
| -port | <port> | Port number (default: 8775) |
| -h, -help | | Show help |

On startup, all configured models are pre-loaded. The web frontend is served at `GET /` from `index.html` in the same directory as the executable.

## 6 REST API

When running `babylon serve`, the following HTTP endpoints are available. All responses include `Access-Control-Allow-Origin: *`.

### 6.1 GET /status

Returns the availability of each engine and the number of loaded voices.

**Response** (`application/json`):

```
{
  "phonemizer": true,
  "kokoro":     true,
  "vits":       false,
  "voices":     54
}
```

`phonemizer`, `kokoro`, and `vits` are booleans indicating whether the model file exists at the configured path. `voices` is the count of `.bin` files in the voices directory.

### 6.2 GET /voices

Returns a sorted JSON array of available Kokoro voice names.

**Response** (`application/json`):

```
["en-GB-alice", "en-US-bella", "en-US-heart", ...]
```

### 6.3 POST /phonemize

Convert text to IPA phonemes or Kokoro token IDs.

**Request body** (`application/json`):

| Field | Type | Required | Description |
|---|---|---|---|
| text | string | Yes | Input text to phonemize |
| tokens | boolean | No | Return token IDs instead of IPA (default: false) |

**Response** — IPA mode:

```
{ "phonemes": "<IPA string>" }
```

**Response** — token mode:

```
{ "tokens": [31, 29, 42, 0, 51, 17, 32, 42] }
```

### 6.4 POST /tts

Synthesise speech. Returns a WAV audio file on success.

**Request body** (`application/json`):

| Field | Type | Required | Description |
|---|---|---|---|
| `text` | string | Yes | Input text |
| `engine` | string | No | `"kokoro"` (default) or `"vits"` |
| `voice` | string | No | Voice name; defaults to the config value |
| `speed` | number | No | Speech speed multiplier (default: `1.0`) |

**Success response**: `audio/wav` binary body.

**Error response** (`application/json`):

```
{ "error": "description of the error" }
```

# 7 C API

Include `babylon.h` and link against `libbabylon`.

## 7.1 G2P (Phonemization)

### 7.1.1 babylon_g2p_init

```
int babylon_g2p_init(const char* model_path,
                     babylon_g2p_options_t options);
```

Loads the Open Phonemizer ONNX model. Returns `0` on success, non-zero on failure.

**Options struct:**

```
typedef struct {
  const char*          dictionary_path; // path to dictionary.json, or
     NULL
  const unsigned char use_punctuation; // 1 to preserve punctuation, 0
     to strip
} babylon_g2p_options_t;
```

### 7.1.2 babylon_g2p

```
char* babylon_g2p(const char* text);
```

Phonemizes `text` and returns a heap-allocated IPA string. The caller must call `free()` on the result.

### 7.1.3 babylon_g2p_tokens

```
int* babylon_g2p_tokens(const char* text);
```

Phonemizes `text` and returns a heap-allocated, `-1`-terminated array of Kokoro-compatible token IDs. The caller must call `free()` on the result.

### 7.1.4 babylon_g2p_free

```
void babylon_g2p_free(void);
```

Releases the G2P session and frees all associated memory.

## 7.2 VITS TTS

### 7.2.1 babylon_tts_init

```
int babylon_tts_init(const char* model_path);
```

Loads a VITS ONNX model. The G2P session must already be initialised. Returns `0` on success.

### 7.2.2 babylon_tts

```
void babylon_tts(const char* text, const char* output_path);
```

Synthesises `text` and writes a WAV file to `output_path`.

### 7.2.3 babylon_tts_free

```
void babylon_tts_free(void);
```

Releases the VITS session.

## 7.3 Kokoro TTS

### 7.3.1 babylon_kokoro_init

```
int babylon_kokoro_init(const char* model_path);
```

Loads the Kokoro ONNX model. The G2P session must already be initialised. Returns `0` on success.

### 7.3.2 babylon_kokoro_tts

```
void babylon_kokoro_tts(const char* text,
                        const char* voice_path,
                        float       speed,
                        const char* output_path);
```

Synthesises `text` using the voice style loaded from `voice_path` at the given `speed`, writing a WAV file to `output_path`.

### 7.3.3 babylon_kokoro_free

```
void babylon_kokoro_free(void);
```

Releases the Kokoro session.

### 7.4 C API Example

```c
#include "babylon.h"
#include <stdlib.h>

int main(void) {
    babylon_g2p_options_t opts = {
        .dictionary_path = "models/dictionary.json",
        .use_punctuation = 1,
    };

    if (babylon_g2p_init("models/open-phonemizer.onnx", opts) != 0)
        return 1;

    if (babylon_kokoro_init("models/kokoro-quantized.onnx") != 0)
        return 1;

    babylon_kokoro_tts(
        "Hello␣world",
        "models/voices/en-US-heart.bin",
        1.0f,
        "output.wav"
    );

    babylon_kokoro_free();
    babylon_g2p_free();
    return 0;
}
```

## 8 C++ API

Include `babylon.h` and link against `libbabylon`. All classes are in their respective namespaces.

### 8.1 OpenPhonemizer::Session

```cpp
namespace OpenPhonemizer {
  class Session {
  public:
    Session(const std::string& model_path,
            const std::string& dictionary_path = "",
            bool use_punctuation = false);

    // Returns concatenated IPA phoneme string for full text
    std::string phonemize(const std::string& text);

    // Returns Kokoro-compatible token IDs
    std::vector<int64_t> phonemize_tokens(const std::string& text);
  };
}
```

## 8.2 Kokoro::Session

```cpp
namespace Kokoro {
  class Session {
  public:
    Session(const std::string& model_path);

    void tts(const std::string& phonemes,
             const std::string& voice_path,
             float speed,
             const std::string& output_path);
  };
}
```

## 8.3 Vits::Session

```cpp
namespace Vits {
  class Session {
  public:
    Session(const std::string& model_path);

    void tts(const std::vector<std::string>& phonemes,
             const std::string& output_path);
  };
}
```

## 8.4 C++ API Example

```cpp
#include "babylon.h"

int main() {
    OpenPhonemizer::Session phonemizer(
        "models/open-phonemizer.onnx",
        "models/dictionary.json",
        /* use_punctuation = */ true
    );

    Kokoro::Session kokoro("models/kokoro-quantized.onnx");

    std::string phonemes = phonemizer.phonemize("Hello␣world");

    kokoro.tts(phonemes,
               "models/voices/en-US-heart.bin",
               /* speed = */ 1.0f,
               "output.wav");

    return 0;
}
```

## 9 Python Wrapper

A pre-built Python package is available as a CI artifact. It bundles the compiled libraries for Linux, macOS, and Windows alongside the Python wrapper module (`babylon.py`).

```python
import babylon

babylon.g2p_init("models/open-phonemizer.onnx",
                 "models/dictionary.json")
babylon.kokoro_init("models/kokoro-quantized.onnx")

babylon.kokoro_tts(
    "Hello␣world",
    "models/voices/en-US-heart.bin",
    speed=1.0,
    output_path="output.wav"
)

babylon.kokoro_free()
babylon.g2p_free()
```

## 10 Web Frontend

When running `babylon serve`, a single-page web interface is served at `http://<host>:<port>/`. It requires no additional dependencies and communicates entirely with the local REST API.

Features:

- **Status indicator** — A dot in the header reflects engine availability from `GET /status`. Cyan indicates at least one engine is ready; red indicates no models are configured.
- **Engine selector** — Switches between Kokoro and VITS. Options are disabled if the corresponding model is not available.
- **Voice selector** — Populated from `GET /voices`; disabled when VITS is selected.
- **Speed slider** — Controls Kokoro speech speed from 0.5× to 2.0×.
- **Phonemize** — Sends `POST /phonemize` and displays the IPA result.
- **Speak** — Sends `POST /tts` and plays the returned WAV audio directly in the browser.
- **Keyboard shortcut** — `Ctrl+Enter` / `Cmd+Enter` triggers speech synthesis.