



> Menu

# Forced Alignment System

Relevant source files

## Purpose and Scope

This document describes WhisperX's forced alignment system, which refines the utterance-level timestamps from ASR transcription to precise word-level and character-level timestamps. The system uses phoneme-based Wav2Vec2 models and CTC (Connectionist Temporal Classification) alignment to map transcribed text to audio waveforms at the frame level.

For information about the ASR transcription stage that precedes alignment, see [Automatic Speech Recognition](#). For information about the overall pipeline orchestration, see [Transcription Pipeline](#).

**Sources:** whisperx/alignment.py | 1-25

## System Overview

The forced alignment system bridges the gap between Whisper's utterance-level timestamps and precise word boundaries. Whisper typically produces segments with timestamps like `[0.0s - 5.2s]` for an entire sentence, but alignment refines this to individual word timestamps like `"hello": [0.0s - 0.3s], "world": [0.4s - 0.8s]`.

## Architecture

Ask Devin about m-bain/whisperX





Sources: [whisperx/alignment.py](#) | 117–389

## Model Selection and Loading

### Supported Languages

WhisperX maintains two model registries for alignment. TorchAudio provides highly optimized models for 5 primary languages, while HuggingFace extends support to 20 additional languages.

Ask Devin about m-bain/whisperX

| Source      | Languages     | Model Type           | Notes                               |
|-------------|---------------|----------------------|-------------------------------------|
| HuggingFace | 30+ languages | Wav2Vec2 checkpoints | Broader coverage, requires download |

Languages with TorchAudio models:

```
DEFAULT_ALIGN_MODELS_TORCH = {
    "en": "WAV2VEC2_ASR_BASE_960H",
    "fr": "VOXPOPULI_ASR_BASE_10K_FR",
    "de": "VOXPOPULI_ASR_BASE_10K_DE",
    "es": "VOXPOPULI_ASR_BASE_10K_ES",
    "it": "VOXPOPULI_ASR_BASE_10K_IT",
}
```

Languages with HuggingFace models (selected examples):

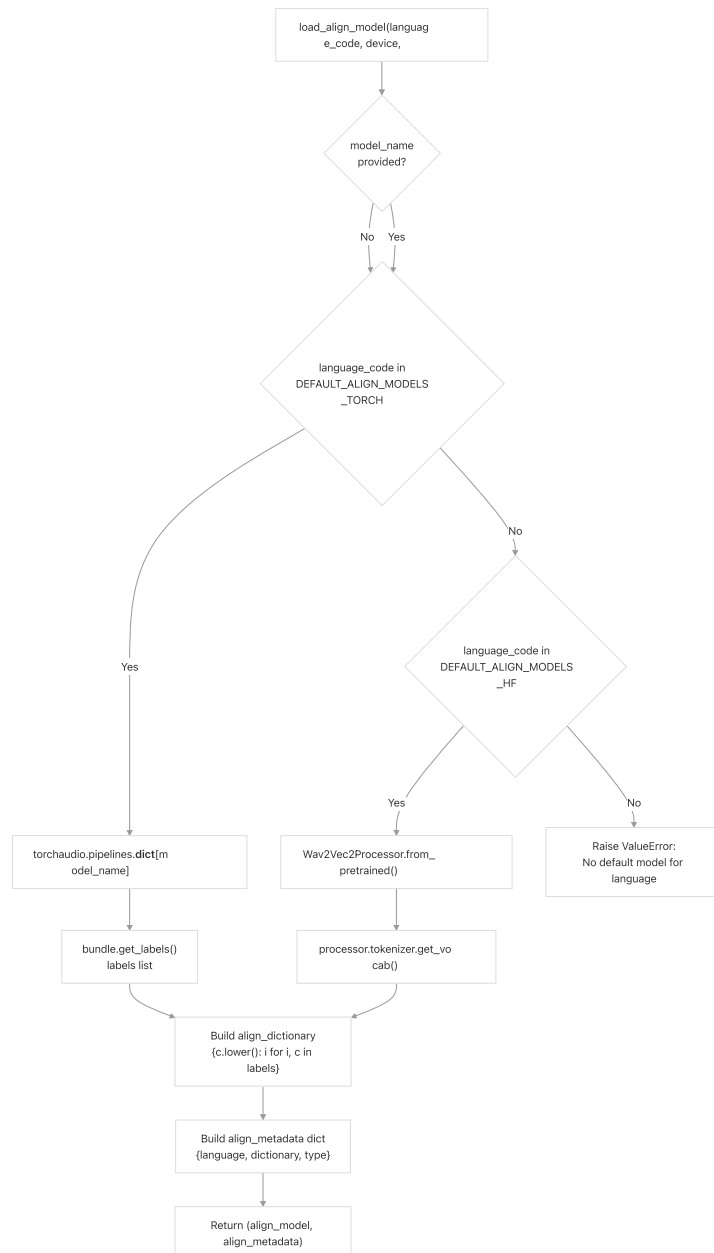
```
DEFAULT_ALIGN_MODELS_HF = {
    "ja": "jonatasgrosmann/wav2vec2-large-xlsr-53-japanese",
    "zh": "jonatasgrosmann/wav2vec2-large-xlsr-53-chinese-zh-cn",
    "ru": "jonatasgrosmann/wav2vec2-large-xlsr-53-russian",
    "ar": "jonatasgrosmann/wav2vec2-large-xlsr-53-arabic",
    # ... 25+ more languages
}
```

Sources: whisperx/alignment.py | 31-77

Model Loading Process

Ask Devin about m-bain/whisperX





The function `load_align_model()` performs model selection and initialization. Key details:

- **TorchAudio path:** Models are accessed via `torchaudio.pipelines.__dict__[model_name]` and loaded with `bundle.get_model()`
- **HuggingFace path:** Models are loaded via `Wav2Vec2Processor` and `Wav2Vec2ForCTC.from_pretrained()`
- **Dictionary construction:** Character-to-index mapping is built from model vocabulary, with all characters lowercased
- **Metadata:** Returns a dictionary with `language`, `dictionary`, and `type` fields for downstream

Ask Devin about m-bain/whisperX

# The Alignment Process

## Function Signature

The primary alignment function has the following signature:

```
def align(  
    transcript: Iterable[SingleSegment],  
    model: torch.nn.Module,  
    align_model_metadata: dict,  
    audio: Union[str, np.ndarray, torch.Tensor],  
    device: str,  
    interpolate_method: str = "nearest",  
    return_char_alignments: bool = False,  
    print_progress: bool = False,  
    combined_progress: bool = False,  
    ) -> AlignedTranscriptionResult
```

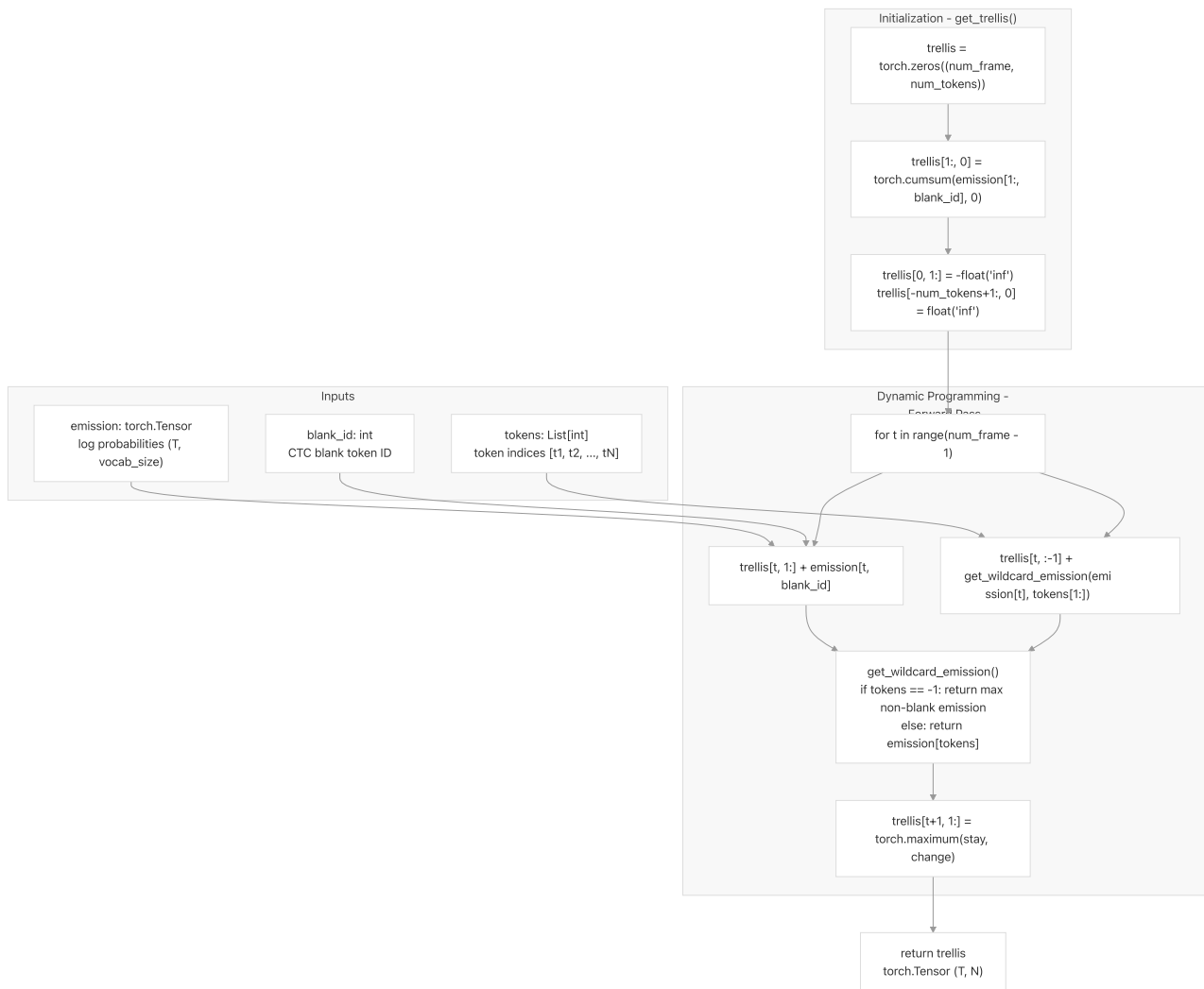
**Sources:** whisperx/alignment.py | 117–127

## Data Flow

Ask Devin about m-bain/whisperX







The `get_trellis()` function implements the forward pass of CTC alignment. At each frame  $t$  and token position  $j$ , it computes the maximum score between:

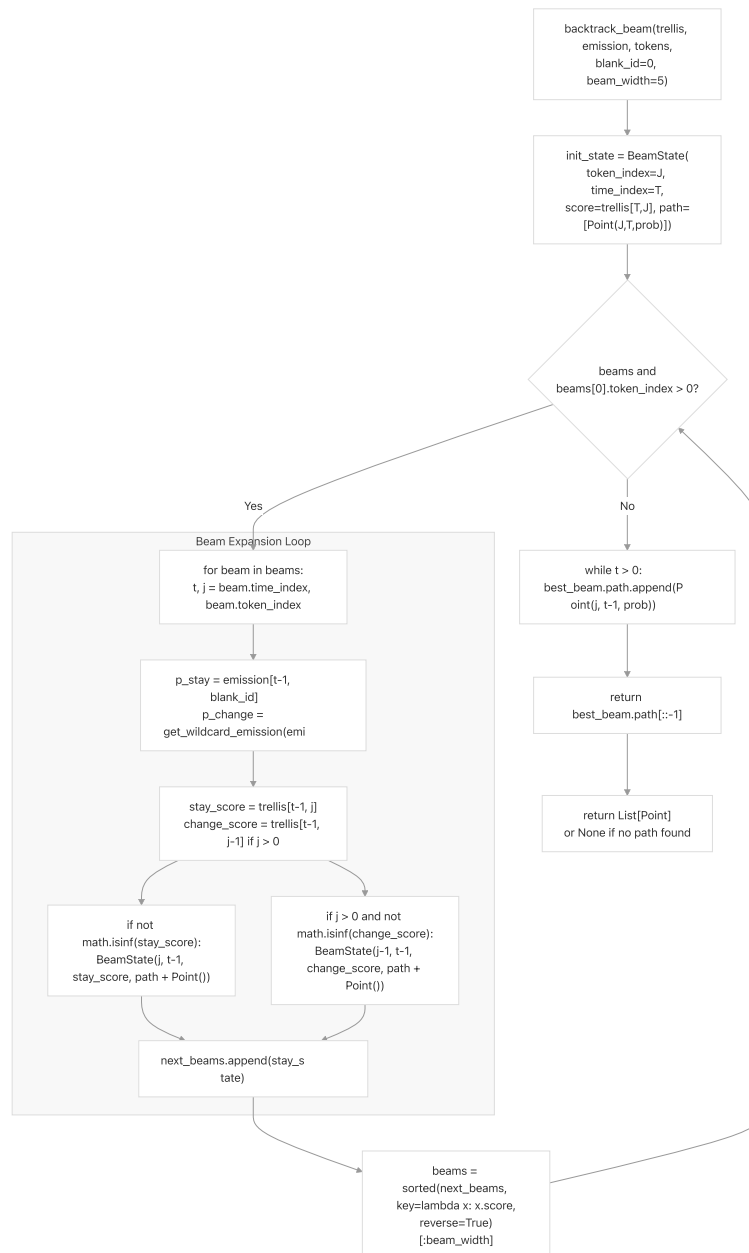
- **Staying** at the same token (emitting blank)
- **Changing** to the next token

**Wildcard handling:** The system supports placeholder tokens (represented as `-1` in the token sequence) for characters not in the model vocabulary. The function `get_wildcard_emission()` assigns the maximum non-blank emission probability to wildcards.

**Sources:** `whisperx/alignment.py` | 395–446

## Beam Search Backtracking

Ask Devin about m-bain/whisperX



The **backtrack\_beam()** function maintains multiple candidate paths (default **beam\_width=2**) during backtracking, selecting the highest-scoring path at each step. Each **BeamState** tracks:

- **token\_index** : Current position in the token sequence
- **time\_index** : Current frame in the audio
- **score** : Cumulative CTC score
- **path** : List of **Point** objects representing the alignment

**Sources:** whisperx/alignment.py | 508–588

Ask Devin about m-bain/whisperX



```
def merge_repeats(path, transcript):
    i1, i2 = 0, 0
    segments = []
    while i1 < len(path):
        # Find all points with same token_index
        while i2 < len(path) and path[i1].token_index == path[i2].token_index:
            i2 += 1
        # Average score and create segment
        score = sum(path[k].score for k in range(i1, i2)) / (i2 - i1)
        segments.append(Segment(
            label=transcript[path[i1].token_index],
            start=path[i1].time_index,
            end=path[i2 - 1].time_index + 1,
            score=score,
        ))
        i1 = i2
    return segments
```

This produces character-level **Segment** objects with **start**, **end**, and **score** attributes representing frame indices in the emission tensor.

**Sources:** whisperx/alignment.py | 590–621

---

## Timestamp Refinement

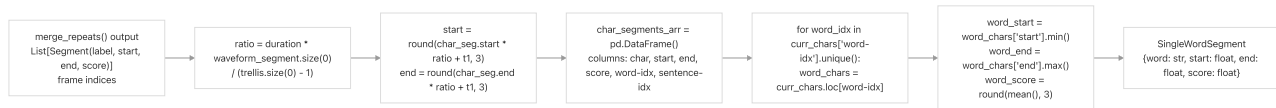
### Character to Word Aggregation

After obtaining character-level alignments, the system aggregates them into word-level timestamps through a multi-step process:

1. **Frame-to-time conversion:** CTC produces frame indices; these are converted to seconds using the ratio **duration / num\_frames**
2. **Character mapping:** Each character in the original text is assigned a timestamp from the aligned character segments
3. **Word grouping:** Characters are grouped by **word-idx**, which increments at word boundaries
4. **Word timestamp computation:** For each word, the system computes:

```
word_start = min(char_start) for non-space characters
```

Ask Devin about m-bain/whisperX



**Sources:** whisperx/alignment.py | 286–355

## Sentence Segmentation

The system uses NLTK's Punkt sentence tokenizer to split segments into sentence-level subsegments. This provides hierarchical structure: segments → sentences → words.

```
# Load language-specific Punkt model
punkt_lang = PUNKT_LANGUAGES.get(model_lang, 'english')
sentence_splitter = nltk_load(f'tokenizers/punkt_tab/{punkt_lang}.pickle')
sentence_spans = list(sentence_splitter.span_tokenize(text))
```

For each sentence span (**start\_idx**, **end\_idx**) :

- Characters within the span are grouped
- Sentence text, start time, and end time are extracted
- Words within the sentence are included in the sentence's word list

**Sources:** whisperx/alignment.py | 195–202    whisperx/alignment.py | 318–361

## Missing Timestamp Interpolation

When alignment fails for certain characters (e.g., due to silence or model limitations), the system uses interpolation to fill missing timestamps:

```
aligned_subsegments["start"] = interpolate_nans(
    aligned_subsegments["start"],
    method=interpolate_method
)
aligned_subsegments["end"] = interpolate_nans(
    aligned_subsegments["end"],
    method=interpolate_method
)
```

Ask Devin about m-bain/whisperX

---

## Language-Specific Handling

### Space vs. Non-Space Languages

WhisperX distinguishes between languages with word-separating spaces and those without:

```
LANGUAGES_WITHOUT_SPACES = ["ja", "zh"]
```

**For space-separated languages** (English, French, German, etc.):

- Text is split by spaces: `text.split(" ")`
- Space character is replaced with `"|"` pipe token (Wav2Vec2 convention)
- Word boundaries are detected by space characters

**For non-space languages** (Japanese, Chinese):

- Each character is treated as a potential word: `per_word = text`
- No space-to-pipe conversion
- Word index increments per character rather than at spaces
- Text aggregation uses `"".join()` instead of `" ".join()`

This handling is pervasive throughout the alignment process:

```
if model_lang not in LANGUAGES_WITHOUT_SPACES:
    per_word = text.split(" ")
    char_ = char_.replace(" ", "|")
else:
    per_word = text # Each char is a word
```

**Sources:** whisperx/alignment.py | 31    whisperx/alignment.py | 161-164

whisperx/alignment.py | 311-314    whisperx/alignment.py | 375-376

### Dictionary Filtering

Characters not present in the model's vocabulary are replaced with placeholder tokens ( `"*"` ) during preprocessing:

Ask Devin about m-bain/whisperX

```

        clean_cdx.append(cdx)
    else:
        clean_char.append('*') # Placeholder
        clean_cdx.append(cdx)

```

These placeholders are later handled by `get_wildcard_emission()`, which assigns them the maximum non-blank emission probability, allowing alignment to proceed even with partial vocabulary coverage.

**Sources:** whisperx/alignment.py | 166–184

---

## Data Schema Transformations

The alignment process transforms data through several schema stages:

### Input Schema

```

# From ASR stage
class SingleSegment(TypedDict):
    start: float      # Utterance start time
    end: float        # Utterance end time
    text: str         # Transcribed text

```

### Intermediate Structures

```

# Preprocessing data (internal)
class SegmentData(TypedDict):
    clean_char: List[str]      # Filtered characters
    clean_cdx: List[int]       # Original indices
    clean_wdx: List[int]       # Word indices
    sentence_spans: List[Tuple[int, int]] # Sentence boundaries

# CTC alignment output (internal)
@dataclass
class Point:
    token_index: int  # Position in token sequence
    time_index: int   # Frame index
    score: float      # Emission probability

```

Ask Devin about m-bain/whisperX

---

score: float # Average probability

## Output Schema

# Word-level result

```
class SingleWordSegment(TypedDict):
    word: str
    start: float
    end: float
    score: float
```

# Aligned segment with words

```
class SingleAlignedSegment(TypedDict):
    start: float
    end: float
    text: str
    words: List[SingleWordSegment]
    chars: Optional[List[SingleCharSegment]] # If return_char_alignments=True
```

# Complete result

```
class AlignedTranscriptionResult(TypedDict):
    segments: List[SingleAlignedSegment]
    word_segments: List[SingleWordSegment]
```

The system converts frame-based alignments (CTC output) to time-based timestamps (user-facing) through the ratio calculation:

$$\text{time\_in\_seconds} = \text{frame\_index} * (\text{segment\_duration} / \text{total\_frames}) + \text{segment\_start}$$

**Sources:** whisperx/schema.py | 1-70

---

## Performance Considerations

### Model Loading Strategy

The alignment models are loaded on-demand per language and explicitly unloaded after use (managed by the orchestrator in **transcribe.py**). This memory management is critical when

Ask Devin about m-bain/whisperX



```
# TODO: Probably can get some speedup gain with batched inference here
waveform_segment = audio[:, f1:f2]
with torch.inference_mode():
    if model_type == "torchaudio":
        emissions, _ = model(waveform_segment.to(device), lengths=lengths)
```

Each segment is aligned independently, which could be optimized for throughput by batching multiple segments together.

**Sources:** whisperx/alignment.py | 248–266

## Minimum Input Length

Wav2Vec2 models require a minimum input length of 400 samples (~25ms at 16kHz). The system pads shorter segments:

```
if waveform_segment.shape[-1] < 400:
    lengths = torch.as_tensor([waveform_segment.shape[-1]]).to(device)
    waveform_segment = torch.nn.functional.pad(
        waveform_segment, (0, 400 - waveform_segment.shape[-1])
    )
```

**Sources:** whisperx/alignment.py | 251–255

## Error Handling

The alignment system includes several fallback mechanisms:

### Alignment Failure Cases

1. **Empty character list:** If no characters in the segment exist in the model dictionary

```
if len(segment_data[sdx]["clean_char"]) == 0:
    logger.warning(f'Failed to align segment (...): no characters found')
    aligned_segments.append(aligned_seg) # Return original timestamps
    continue
```

Ask Devin about m-bain/whisperX



```
aligned_segments.append(aligned_seg)
    continue
```

### 3. **Backtrack failure:** If CTC backtracking cannot find a valid path

```
path = backtrack_beam(trellis, emission, tokens, blank_id, beam_width=2)
if path is None:
    logger.warning(f'Failed to align segment (...): backtrack failed')
```

Ask Devin about m-bain/whisperX

