ChatGPT

Node.js/TypeScript Sentiment Analysis Pipeline (Short-Video Transcripts)

Building an in-memory sentiment analysis prototype for TikTok/Reels/Shorts transcripts means choosing accurate, easy-to-integrate tools. Major cloud NLP services provide robust sentiment APIs: for example, **AWS Comprehend** can classify text as *POSITIVE*, *NEGATIVE*, *NEUTRAL*, or *MIXED* [1]; **Google Cloud Natural Language** and **Azure Cognitive Services (Text Analytics)** similarly return sentiment labels and scores [2] [3]. These services offer official Node.js SDKs or REST endpoints for quick integration. OpenAI's GPT models (via API) can also analyze sentiment in context, excelling at nuance [4]. For a simpler local approach, NPM libraries like **wink-sentiment** (AFINN-based) or **sentiment** can score text immediately. For instance, *wink-sentiment* reports ~77% accuracy on product reviews [5], but lexicon-based tools generally handle context and slang less well than ML models.

- **AWS Comprehend (Sentiment API):** Uses ML to label text as POSITIVE/NEGATIVE/NEUTRAL/MIXED with confidence scores [1]. (Free tier: 5M chars/month [6].)
- **Google Cloud Natural Language:** Sentiment API returns document- and sentence-level sentiment (score and magnitude) [7]. (5M chars free/month; ~$0.001 per 1000 chars beyond [8].)
- **Azure Text Analytics (Language API):** Provides overall and sentence-level sentiment labels and scores [3]. (5,000 text records free/month [9].)
- **OpenAI GPT-3.5/4 API:** Can be prompted to analyze sentiment, capturing complex or sarcastic tones [4]. (Cost: fractions of a cent per request, very high accuracy.)
- **Node Libraries:** Tools like `wink-sentiment` or `sentiment` run offline, returning a score per phrase or sentence (fast and free but less nuanced than cloud models [5]).

Splitting Transcripts into Sentences

Accurate sentiment often requires meaningful sentence or phrase boundaries. If your transcript is raw text (timestamps removed), you can split on punctuation or use language-aware tokenization. In modern Node.js (v20+), the built-in `Intl.Segmenter` can split text by sentence using locale rules: e.g. `new Intl.Segmenter('en', { granularity: 'sentence' })` segments text into sentences [10]. Alternatively, use an npm package like `sbd` which applies rule-based sentence splitting ("~95% accuracy" by skipping abbreviations, etc [11]). For a quick hack, a regex such as `text.split(/[.?!]\s+/)` or splitting on newlines may suffice for short, informal transcripts. In code, you might do:

- **Intl.Segmenter:**

```
const segmenter = new Intl.Segmenter('en', { granularity: 'sentence' });
const sentences = Array.from(segmenter.segment(transcriptText), s =>
s.segment);
```

- **sbd library:**

```
import { sentences } from 'sbd';
const sentences = sentences(transcriptText, { newline_boundaries: true });
```

Each resulting element can then be analyzed individually.

Sample Implementation Flow

1. Fetch and Clean Transcript: Use a HTTP client (e.g. `axios` or `fetch`) to call your transcript API (e.g. ScrapeCreators). Remove timestamps or artifacts with a regex, e.g. `text.replace(/`

$$\d1,2:\d2$$

`/g, '')`.

2. Split into Sentences: Segment the cleaned text as above. This yields an array of phrases/sentences.

3. Set Up Sentiment Client: Choose your analysis tool. For example, with AWS Comprehend:

```
import { ComprehendClient, DetectSentimentCommand } from "@aws-sdk/client-comprehend";

const client = new ComprehendClient({ region: "us-east-1" });  // configure AWS creds
```

Or with a local library:

```
import Sentiment from 'sentiment';
const sentiment = new Sentiment();
```

4. Analyze Each Sentence: Iterate sentences and call the API or library. For AWS Comprehend:

```
const results = await Promise.all(sentences.map(async sentence => {
  const cmd = new DetectSentimentCommand({
    LanguageCode: "en", Text: sentence
  });
  const res = await client.send(cmd);
  return { text: sentence, sentiment: res.Sentiment, score:
```

```
res.SentimentScore };
}));
```

(Comprehend returns `res.Sentiment` in {POSITIVE, NEGATIVE, NEUTRAL, MIXED} [1] with confidence scores.) For a library:

```
const results = sentences.map(s => {
  const { score } = sentiment.analyze(s);
  return { text: s, score };
});
```

5. Compute Overall Sentiment: Combine sentence-level results for a video-level label. A simple approach is to average or vote on positive vs negative scores. For example, count how many sentences are positive vs negative to pick an overall label, or average the confidence scores. (Azure's API directly returns an overall document score with confidence if you prefer a single call [3].)

6. Return/Output Results: Since the app is in-memory, you can immediately use or return the `results` array and overall label without saving to disk. For example, return a JSON `{ videoSentiment: "Positive", sentenceSentiments: results }`.

Performance, Cost & Limitations

For an MVP, local analysis runs nearly instantly (millisecond-range) and costs nothing, but may miss nuance (AFINN-based tools ignore context like sarcasm). Cloud APIs incur network latency (~100–200 ms per request) and per-use fees, but offer higher accuracy and context. In practice, processing a short transcript (a few hundred words) with AWS/Google/Azure will take only a few hundred milliseconds total. Most services have generous free tiers: AWS and Google offer ~5 million free characters per month [6] [8], Azure includes ~5,000 free text records [9]. After that, costs are on the order of $0.0001–$0.001 per 1000 characters (i.e. fractions of a cent per video) [6] [8]. LLM calls (OpenAI) cost a few cents per thousand tokens.

Limitations: This prototype assumes **English** transcripts only (non-English input could be detected or filtered out). Sentiment analysis also relies purely on text – it may not catch intonation or visual cues in videos. Short-form videos often use slang, emojis, or mixed emotions, which can challenge simple models. Finally, doing all work in-memory means you cannot query past analyses – each transcript is analyzed anew. Despite these caveats, this approach yields a fast, first-pass sentiment solution that's easy to integrate and iterate on.

Sources: Popular sentiment APIs and NLP libraries [2] [3] [4]; Intl.Segmenter and sentence-splitting tools [10] [11]; AWS Comprehend pricing/free-tier [6] and Google Cloud pricing [8].

---

[1] DetectSentiment - Amazon Comprehend API Reference
https://docs.aws.amazon.com/comprehend/latest/APIReference/API_DetectSentiment.html

2 3 4 7 Best Sentiment Analysis APIs in 2025 | Eden AI
https://www.edenai.co/post/best-sentiment-analysis-apis

5 wink-sentiment - npm
https://www.npmjs.com/package/wink-sentiment

6 Natural Language Processing – Amazon Comprehend Pricing – AWS
https://aws.amazon.com/comprehend/pricing/

8 Pricing | Cloud Natural Language | Google Cloud
https://cloud.google.com/natural-language/pricing

9 Explore Free Azure Services | Microsoft Azure
https://azure.microsoft.com/en-us/pricing/free-services

10 Intl.Segmenter - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/Segmenter

11 sbd - npm
https://www.npmjs.com/package/sbd