NATURAL LANGUAGE PROCESSING IN AI-DRIVEN CHATBOTS: From Text Preprocessing to Context-Aware Responses

Shruti Malik
ITEC5025 – Artificial Intelligence
Week 7 Assignment: Data Collection and Preprocessing
February 24, 2026

Introduction

When I first started working on this chatbot project for ITEC5025, I quickly realized that building a bot that could actually understand what someone is trying to say—rather than just matching keywords—was a lot harder than it looked. That gap between what a user types and what a computer can parse is exactly what Natural Language Processing (NLP) is designed to close. In this paper, I will walk through the NLP concepts that have shaped how I think about building an intelligent, context-aware clinical chatbot: from why NLP matters in the first place, all the way through word embeddings, training strategies, and the design decisions behind making a chatbot that actually remembers what you just told it.

1. Defining Natural Language Processing and Its Importance to AI-Driven Chatbots

Natural Language Processing, or NLP, is a branch of artificial intelligence that focuses on enabling computers to read, understand, and generate human language in a meaningful way (Jurafsky & Martin, 2023). Human language is wonderfully messy—we abbreviate, we use slang, we rely heavily on context, and we often mean something entirely different from what we literally say. NLP gives machines the tools to navigate all of that.

For AI-driven chatbots, NLP is not just a nice-to-have feature—it is the entire foundation. Without NLP, a chatbot can only respond to exact keyword matches, which means a user who types “show me info on patient P000001” gets a different result than someone who types “get patient P000001,” even though they want the same thing. That kind of fragility makes for a frustrating user experience. With NLP, the bot can understand that both phrases share the same intent, regardless of surface-level differences.

In building the Hypotify Clinical Chatbot for this course, NLP has allowed the system to accept natural language queries instead of rigid command syntax; detect the user’s emotional state through sentiment analysis; extract structured information like patient IDs and diagnoses from free text; and maintain conversational context across multiple turns. These capabilities transform the chatbot from a glorified search box into something that genuinely feels like a conversation.

The clinical domain makes NLP especially important. Patients and healthcare providers do not always use precise medical terminology—someone might type “chest tightness” instead of “angina,” or ask about “blood sugar” instead of “glucose level.” NLP bridges that vocabulary gap, making the chatbot more accessible and more useful in real-world clinical settings.

2. Advanced NLP Techniques: Sentiment Analysis and Named Entity Recognition

Beyond basic intent classification, two NLP techniques dramatically expand what a chatbot can understand: sentiment analysis and Named Entity Recognition (NER). I found both of these particularly interesting to implement, because they operate on entirely different dimensions of language.

Sentiment Analysis

Sentiment analysis is the process of automatically identifying the emotional tone of a piece of text—whether it is positive, negative, or neutral (Liu, 2022). In the context of a clinical chatbot, this capability is surprisingly important. A message like “I’ve been feeling really hopeless lately” carries very different implications than “I feel great, no complaints.” A sentimentally-aware chatbot can detect distress signals and respond with empathy, flag the conversation for human escalation, or adjust its tone accordingly.

For the Week 7 chatbot, I implemented sentiment analysis using NLTK’s VADER (Valence Aware Dictionary and sEntiment Reasoner) tool. VADER is particularly well-suited for conversational text because it was designed to handle the informal, often punctuation-heavy style of human messages. When a user types “sentiment I’m really worried about my results,” the chatbot runs the text through VADER’s scoring algorithm, which returns a compound score ranging from −1.0 (maximally negative) to +1.0 (maximally positive). A score below −0.05 triggers a response that acknowledges the user’s concern.

Named Entity Recognition (NER)

Named Entity Recognition is the task of automatically detecting and categorizing structured entities within unstructured text (Nadeau & Sekine, 2007). In plain English: it answers the question “who, what, when, and where is mentioned in this sentence?”

For a clinical chatbot, NER is enormously valuable. When a user types a message like “Patient P000123 was admitted with hypertension and a glucose level of 142 on 2025-03-15,” NER can extract: the patient ID (P000123), the clinical condition (hypertension), the lab value (glucose 142), and the date (2025-03-15). This transforms free-form user input into structured, actionable data that can be queried against the patient database.

In the Hypotify chatbot, I implemented NER using a hybrid approach: regex patterns for highly-structured entities like patient IDs (e.g., P followed by six digits) and dates, and

NLTK's built-in chunker for person names. This combination is lightweight but effective for the clinical domain, where many of the most important entities follow predictable formats.

Together, sentiment analysis and NER give the chatbot two critical "senses"—emotional intelligence and information extraction—that a keyword-matching system simply cannot provide.

3. Text Preprocessing and Tokenization in NLP

Before any NLP model can work with human language, the raw text has to be cleaned and structured in a way the computer can process. This is where text preprocessing comes in, and in my experience building this chatbot, it is one of those foundational steps that is easy to overlook but absolutely critical to get right.

Why Preprocessing Matters

Raw text from users is inconsistent by nature. People mix uppercase and lowercase, use punctuation inconsistently, include typos, and employ contractions that would confuse a naïve word-matching system. Preprocessing normalizes all of this so the model sees clean, consistent input. The preprocessing pipeline in the Week 7 chatbot follows these steps:

1. **Lowercasing:** "Hello" and "hello" and "HELLO" all become "hello." This simple step significantly reduces vocabulary size.
2. **Tokenization:** The lowercased text is split into individual words using NLTK's *word_tokenize()*. Tokenization is the bridge between raw text and numerical representations.
3. **Stop word removal:** Common words like "the," "a," and "me" carry little semantic meaning. Filtering them lets the model focus on words that matter.

4. **Stemming:** Using NLTK's PorterStemmer, words are reduced to their root form. "Running," "runs," and "ran" all become "run."

Tokenization in the Chatbot Context

For chatbots, tokenization serves two roles. First, at training time, it converts the sample patterns in *intents.json* into preprocessed token lists that the Keras Tokenizer can convert to integer sequences. Second, at inference time, it transforms the user's live input through the same pipeline before feeding it to the model, ensuring the same vocabulary mapping is applied consistently.

A mismatch between training-time and inference-time preprocessing is one of the most common—and hardest to debug—sources of poor chatbot performance. By using the same preprocessing functions across both stages (as done in *train_model.py* and *chatbot_model.py*), the Hypotify chatbot avoids this pitfall.

4. Word Embeddings and the Continuous Vector Space

The Old Way: One-Hot Encoding

The naive approach is one-hot encoding: each word in the vocabulary gets its own vector of zeros with a single one in the corresponding position. A vocabulary of 10,000 words produces 10,000-dimensional vectors where every pair of words is equidistant from every other pair. "King" and "Queen" are just as far apart as "King" and "banana." This representation is computationally expensive and semantically meaningless.

Word Embeddings: A Better Representation

Word embeddings represent each word as a dense, low-dimensional vector (typically 50–300 dimensions) in a continuous vector space, where the position of each word encodes its meaning (Mikolov et al., 2013). Words with similar meanings end up in similar positions in the vector space. Words that appear in similar contexts—"doctor" and "physician," or "happy" and "joyful"—get pulled toward each other during training.

This enables vector arithmetic on meaning. The famous example is:

$$vector(\text{“king”}) - vector(\text{“man”}) + vector(\text{“woman”}) \approx vector(\text{“queen”})$$

The model has learned that "king" and "queen" are related along a royalty dimension, and "man" and "woman" differ along a gender dimension, without ever being explicitly told any of this.

Why This Matters for Chatbots

For a clinical chatbot, word embeddings mean that "patient lookup" and "show patient record" will map to nearby positions in the embedding space, even if neither phrase appeared in training data. This generalization is what separates an embedding-based chatbot from a simple keyword matcher.

In the Week 7 chatbot, the Embedding layer in the Keras model learns 64-dimensional word representations directly from the training data. These are tailored to our specific domain vocabulary rather than using generic pre-trained representations.

5. Word Embedding Techniques: Word2Vec, GloVe, and FastText

Word2Vec

Word2Vec, introduced by Mikolov et al. (2013) at Google, learns word embeddings by training a shallow neural network on two prediction tasks: Continuous Bag of Words (CBOW) predicts a word from its context; Skip-gram predicts context words from a center word. By optimizing these predictions, the model incidentally places semantically similar words near each other in the vector space. **Strengths:** Fast to train; high-quality for large corpora. **Weaknesses:** Cannot handle out-of-vocabulary (OOV) words; no morphological awareness.

GloVe (Global Vectors for Word Representation)

GloVe, developed at Stanford (Pennington et al., 2014), uses global co-occurrence statistics rather than a predictive model. It builds a matrix of how often each word co-occurs with every other across a large corpus, then factorizes it to produce dense vectors. **Strengths:** Strong on analogy tasks; efficient single-pass training. **Weaknesses:** Struggles with OOV words; pre-trained vectors may encode corpus biases.

FastText

FastText, developed by Facebook AI Research (Joulin et al., 2016), represents words as bags of character n-grams. The embedding for “running” is built from its subwords: “run,” “runn,” “unni,” etc. **Strengths:** Handles OOV words gracefully—critical for clinical text with abbreviations and misspellings. **Weaknesses:** Larger models; more implementation complexity.

For the Week 7 Hypotify chatbot, a learned Embedding layer trained from scratch on our intent patterns was used rather than pre-trained embeddings, because our vocabulary is highly domain-specific and our dataset is small. Pre-trained embeddings like GloVe or FastText would be the right choice when scaling to a larger, more open-ended conversational dataset.

6. Training Word Embeddings on a Dataset: Context and Semantic Relationships

The Role of Context

The distributional hypothesis in linguistics holds that “a word is characterized by the company it keeps” (Firth, 1957, as cited in Jurafsky & Martin, 2023). Words that appear in similar contexts tend to have similar meanings—this is the theoretical foundation for all word embedding methods.

During training, the model slides a window across a corpus, examining each word alongside its neighbors. For each example, it asks: “Given this context, how likely is this center word?” (CBOW) or “Given this word, how likely are these context words?” (Skip-gram). Repeatedly adjusting the embedding vectors to improve these predictions produces a representation where context-similar words are vector-similar.

How Semantic Relationships Are Captured

The training process captures several relationship types simultaneously:

- **Synonymy:** “happy” and “joyful” appear in similar contexts → similar vectors.
- **Analogy:** “Paris” relates to “France” as “Berlin” relates to “Germany”—the displacement vector is consistent.
- **Domain relations:** In a clinical corpus, “hypertension” and “blood pressure” cluster together because they co-occur with the same medical context words.

In the Hypotify chatbot training pipeline (*train_model.py*), the Embedding layer is initialized with random weights and updated via backpropagation during model training. After 200+ training epochs, embedding vectors for “show,” “display,” and “view” will be close together in the 64-dimensional space, because all three words appear in the context of the

patient_lookup intent.

7. Context-Aware Responses in Chatbots and the User Experience

Here is something that bothered me early in this project: the Week 5 chatbot was smart about understanding single commands, but it had no memory. Every message was treated as if it were the first. Ask it about a patient, then follow up with “tell me more about that,” and it would have no idea what “that” referred to. That experience—talking to something with no memory—feels uncanny and frustrating in a way that is hard to explain until you experience it.

Context-aware responses solve this problem. A context-aware chatbot remembers recent conversation turns and uses that history to interpret ambiguous messages, avoid repeating information, and produce responses that feel coherent over the course of a conversation.

How Context-Awareness Enhances the User Experience

1. **Pronoun and reference resolution:** When a user asks “what is her diagnosis?” after discussing a specific patient, the chatbot resolves “her” to the last mentioned patient record.
2. **Follow-up handling:** A user who received a list of Spanish-speaking patients and then types “show me the first one” can be understood only if the chatbot knows what was just displayed.
3. **Emotional continuity:** If a user expressed frustration two turns ago, a context-aware chatbot can acknowledge it, making the interaction feel more human.
4. **Avoiding repetition:** Context lets the chatbot recognize when something has already been said, preventing redundant responses.

In clinical settings these benefits are especially pronounced. Healthcare interactions are often complex and emotionally charged. A chatbot that "forgets" what was just discussed wastes the clinician's time and, in patient-facing applications, can feel dismissive or even alarming.

8. Strategies and Techniques for Context-Aware Chatbot Responses

Making a chatbot truly context-aware requires deliberate design across several dimensions. In building the Week 7 chatbot, five concrete strategies were implemented:

Strategy 1: Conversation History Window

The *ConversationContext* class stores the last five turns as a deque (a memory-efficient circular buffer). Each entry records the user's message, the detected intent, the chatbot's response, and a timestamp. Five turns balances multi-turn reference tracking against inappropriate influence from early-session context.

Strategy 2: Last Intent Tracking

The class tracks the last detected intent, enabling the *is_followup()* method to detect phrases like "tell me more" or "elaborate" and route them back to the previous topic automatically.

Strategy 3: Entity Persistence

The *last_entity* attribute persists the most recently mentioned patient ID or resource, so follow-up questions like "what language does she speak?" can be answered without requiring the user to repeat the ID.

Strategy 4: Confidence-Based Fallback

The TF intent classifier assigns a confidence score to every prediction. When confidence falls below 0.50, the chatbot politely asks for clarification rather than guessing—one of the fastest ways to destroy user trust is a confidently wrong response.

Strategy 5: Keyword-First Routing

For predictable, high-frequency commands like “hello,” “help,” and “goodbye,” the chatbot uses a fast in-memory keyword lookup (mirroring the Week 6 SQL *chatbot_responses* table) before invoking the TF model, reducing latency for common interactions.

Conclusion

Building the Hypotify Clinical Chatbot across Weeks 5, 6, and 7 has given me a much deeper appreciation for what makes conversational AI hard. Text preprocessing, tokenization, word embeddings, and context management are not just academic concepts—they are practical engineering decisions with real consequences for how useful and trustworthy the final system is.

The shift from keyword-matching (Week 5) to embedding-based TensorFlow intent classification (Week 7) is a microcosm of the broader evolution of NLP over the past decade. The underlying goal has not changed: help computers understand human language. But the tools available to achieve that goal have improved dramatically, and the strategies explored in this paper represent both the current best practices and the conceptual foundations for understanding where NLP will go next.

References

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*. https://arxiv.org/abs/1607.01759

Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed. draft). Stanford University. https://web.stanford.edu/~jurafsky/slp3/

Liu, B. (2022). *Sentiment analysis and opinion mining* (2nd ed.). Morgan & Claypool Publishers. https://doi.org/10.2200/S01397ED2V01Y202112HLT063

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. https://arxiv.org/abs/1301.3781

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes, 30*(1), 3–26. https://doi.org/10.1075/li.30.1.03nad

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162