Title? Author? Year?

Chatbots In Your Face For Anybody

Andrew Fowlie

2026

Andrew Fowlie is human and can make mistakes.

Ask anything

Contents

1. Past

2. Representing text

3. Analysing text

4. Predicting text

Section 1

Past

The imitation game

ELIZA

```
USER: I'd like to talk.
ELIZA: Please tell me more.
USER: I'm feeling great.
ELIZA: Can you say a bit more about that?
USER: I don't know.
ELIZA: Why do you say that?
```

ELIZA was developed by Joseph Weizenbaum in 1966 as an empathetic psychotherapist. Try it at `https://elizaemulator.com`

Naming of ELIZA was gendered and female — named after character from *Pygmalion*

ELIZA was rule based

Rule based

User input ⇒ Apply rules ⇒ ELIZA output

- Match **keywords** in your input— looks for e.g., *mother*, *I feel*, *because*, or *I can't* and then uses a matching rule
- If nothing matches, falls back to e.g., *please tell me more*
- **Transforms** your input to output using that rule — e.g., turns it into a plausible question
- **Open-ended** — keeps you talking by asking open-ended questions
- **Stateless** — it's a *pure* function — it just applies a rule and doesn't update any internal state

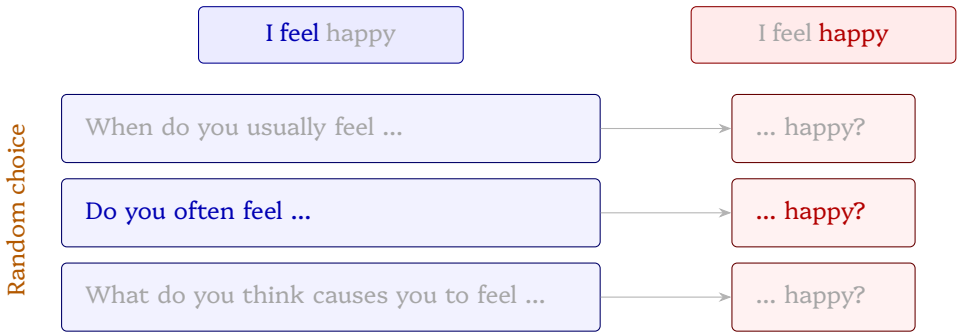
I feel happy, ELIZA

```
USER: I feel happy
```


```
ELIZA: Do you often feel happy?
```

I feel happy, ELIZA

```
USER: I feel happy
```




```
ELIZA: Do you often feel happy?
```

Why does this work?

- We do mirror vocabulary in conversations, to an extent
- We do use *memorized* or *stock* responses, sometimes
  - How are you?
  - I'm fine thanks
  - I'm sorry to hear that
  - That sounds ...
- **ELIZA-effect** — we project human traits onto even basic computer programs
- Does ELIZA feel human to you?
- Probably not, but it does show that simple rules and patterns can imitate human conversation

PARRY

- ELIZA was cool, but its responses were generic
- It didn't get mad or upset or happy with you
- What if we gave it an **emotional state**?
- In 1972 Kenneth Colby invented **PARRY** as a computer program that simulated a paranoid schizophrenic
- The emotional state was described by three variables
    1. Fear
    2. Anger
    3. Distrust

Where were you, PARRY?

```
USER: Where were you last night?
```

Where were you last night?

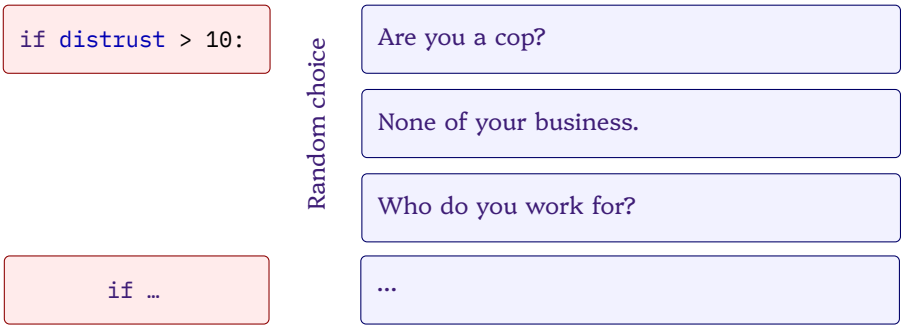
Update state

```
fear = fear + 2
```

```
distrust = distrust + 1
```

```
anger = anger + 1
```

Where were you, PARRY?




```
PARRY: Are you a cop?
```

Why does this work?

- **Passed** Turing test — psychiatrists found transcripts from PARRY indistinguishable from those from real patients
- More complicated than ELIZA — rules *and* state
- However, it imitated a narrow slice of human conversation — responses from a paranoid schizophrenic patient being interviewed by a psychiatrist
- As with ELIZA, this isn't general chatting
- Although it passed the Turing test, PARRY could easily be distinguished from patients: PARRY only had an **emotional state** — it didn't have a **contextual state**
- It could never e.g. repeat anything back to you

RACTER

- RACTER was a story-telling chatbot developed in early 1980s by William Chamberlain and Thomas Etter to write science fiction stories
- Short for *raconteur* — due to programming constraints, program names could only be six characters long
- Published in 1984 by MindScape — a video-game developer — for home computers such as the Apple II
- Like PARRY, it used rules and state, but it took it *much* further

RACTER

Rather than a list of numbers, the state captured rich **contextual information**, e.g., the topics recently discussed or a list of recent proper nouns

Example of state

Scalar

```
fear = 11
anger = 2
mistrust = 7
```

Symbolic

```
recent_topics = [football, money]
recent_proper_nouns = [Ronaldo, Messi]
recent_verbs = [play]
```

Tell a story, RACTER

- The rich contextual state was constructed by searching for keywords in user input using search rules
- That state was combined with user input using an elaborate set of rules and templates
- Did it work? You be the judge!

```
More than iron, more than lead, more than gold I need electricity.
I need it more than I need lamb or pork or lettuce or cucumber.
I need it for my dreams.
```

RACTER, *The Policeman's Beard Is Half Constructed* (1983)

Jabberwacky

- Rule-based chatbots were turbocharged in the 90s by adding more and more rules, e.g., **ALICE**, but no conceptual breakthrough
- **JABBERWACKY** (1997) was different: it was **example and retrieval** based
- It used a ***database*** of examples of inputs and outputs
- It found the ***best*** match your input in its examples, and ***retrieved*** the corresponding output
- It had heuristic rules to find the ***best*** match, that could include context e.g. previous inputs and outputs

Why did it work?

Built a database of examples — millions of entries — from chatting to users

```
JABBERWACKY: What did you do yesterday?
USER: Nothing interesting.  I was working all day.
```

Record this in database as an input/output example. Next user ...

```
USER: Did you do anything yesterday?
```

Retrieve best match: What did you do yesterday?

```
JABBERWACKY: Nothing interesting.  I was working all day.
```

WOAH! Modern methods

- Until even 2010, chatbots were mostly seen as **silly and humorous**
- Machine intelligence — e.g., chess playing programs that beat Garry Kasparov — was seen as **serious and the route towards artificial intelligence**
- That was about to change as several fields collided: chatbots, machine translation, and machine intelligence
- The key lesson: explicit rules and an internal state only get you so far; **training on millions of examples beats everything else**
- We'll now jump forward to modern large-language models

Section 2

Representing text

Tokenization

When looking at user input, what are the most natural units?

Words — `What` | `␣` | `did` | `␣` | `you` | `␣` | `do` | `␣` | `yesterday` | `?`

- In English there are more than 500k words
- How would we treat words we hadn't seen before?
- How would we *traet* typos?

Tokenization

When looking at user input, what are the most natural units?

Characters — W|h|a|t|□|d|i|d|□|y|o|u|□|d|o|□|y|e|s|t|e|r|d|a|y|?

- Only 26 letters + capitals + punctuation — about 100
- Individual characters carry little **semantic** or **syntactic** information
- Sequences are long when expressed in characters — *What did you do yesterday?* is 5 words but 26 characters

Tokenization

When looking at user input, what are the most natural units?

Syllables — `What` | □ | `did` | □ | `you` | □ | `do` | □ | `yes` | `ter` | `day` | `?`

- Connected to **phonetics** — how things sound
- Not obviously units of **meaning**
- Syllables depend on pronunciation

Tokens — What□|did□|you□|do□|yester|day|?

- LLMs use **tokens** as the building blocks to represent text
- Tokens needn't be whole words and can include spaces and punctuation
- To compute efficiently, we must balance **vocabulary size** and **sequence length**
  - Whole words + punctuation — enormous vocabulary and short sequence length
  - Characters + punctuation — tiny vocabulary but long sequence length
- Special **tokenizer** algorithms find the sweet spot — compromise driven by computation, not linguistics
- Explore tokenizers at `https://gpt-tokenizer.dev/`

Vectorization

- We've broken down text into tokens using our tokenizer; tokens are *still text though*
- How can we represent tokens and quantify their semantic relationships?
- We don't want `cat` ≈ `car`; although their **letters** are similar, cats are not like cars
- We do want `cat` ≈ `dog` as they are often **interchangeable** in sentences
- We represent tokens by coordinates on a map — `cat` and `dog` are nearby on the map

The Map

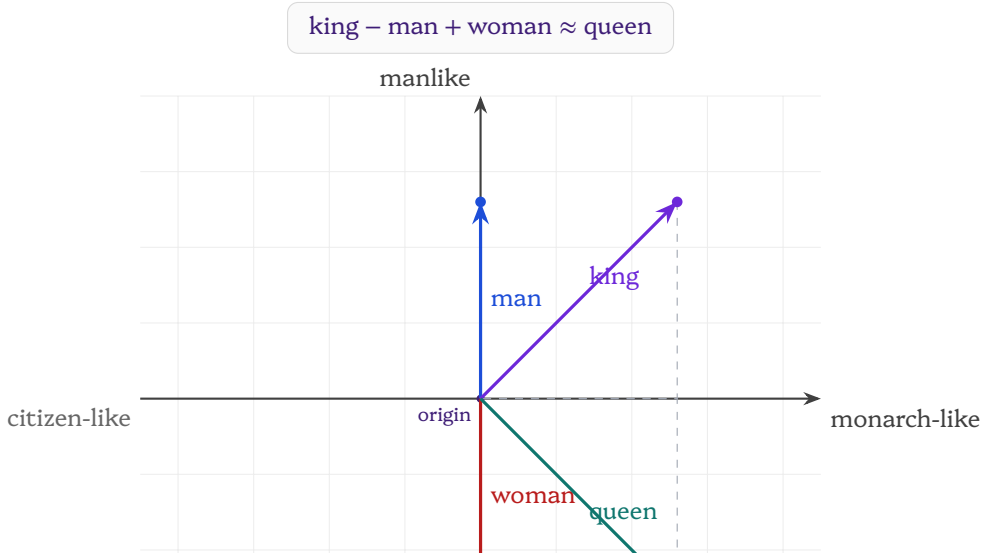
- The map isn't a two-dimensional atlas or street map; it's a **high-dimensional** mathematical space called the **embedding space**
- On a map, we represent a place by two numbers — latitude and longitude in an atlas or e.g. square C7 on a street map
- E.g., Rome is represented by two numbers, latitude and longitude

```
Rome = (41.89193, 12.51133)
```

- In an embedding space, we represent a token by **several numbers**, one for each dimension
- E.g., `cat` is represented by n numbers in an n-dimensional embedding space

```
cat = (0.343, 0.121, -0.982, …, 0.754)
```

Vectorization



Choosing the vectors

- Rome's latitude and longitude were set by Rome's location
- What sets the size of the embedding space?
- The size of the embedding space is a **hyperparameter** chosen to balance
  - big enough to capture semantics of many different tokens
  - small enough to be managable on current computers
- What sets the vector that represents `cat`?
- These vectors are **parameters** that are chosen by
  - trying many different choices, and seeing what works best
  - this uses training data; we'll come to this later

Section 3

Analysing text

Attention

Section 4

Predicting text

Feed forward NN

Output layer

References