

Presenting...
Prompt Engineering in Emacs

Shane Mulligan

<2021-03-01 Mon>

Following along

Repositories for following along

```
http://github1s.com/semiosis/exemplary  
http://github1s.com/semiosis/pen.el  
http://github1s.com/semiosis/prompts  
http://github1s.com/semiosis/prompt-engineering-patterns  
http://github1s.com/minimaxir/gpt-3-client
```

slides

```
http://github1s.com/semiosis/presentation-prompt-engineering-in-emacs
```

glossary

```
http://github1s.com/mullikine/glossaries-gh/blob/master/openai.txt
```

Demo

```
1  ssh -oBatchMode=no shane@124.197.60.232 -p 9922
```

Text Generator

Background knowledge

- GPT-3 is a seq2seq model (a text generator)
 - It's stochastic but can be configured to be deterministic.

Key concepts

- prompt,
- completion, and
- tokens

Limitations

Combined, the text prompt and generated completion must be below 2048 tokens (roughly ~1500 words).

context-stuffing With only 2048 tokens, you need to make use of your real estate by providing instructions and making implicit information explicit.

Characteristics

- declarative, like `html`
- stochastic, like `problog`
- Unlocks new types of applications
- Speeds up development

Some prompts I've made

generate-vim-command.prompt

```
1 Vim
2
3 Insert "Q: " at the start of the line
4 :%s/^/Q: /g.
5 ###
6 Remove whitespace from the start of each line
7 :%s/^\s*/\1/g
8 ###
9 Join each line with the next line
10 :1,$j
11 ###
12 Make all occurrences of Steve lowercase
13 :%s/Steve/steve/g
14 ###
15 <1>
```

Classification

- Tweet Sentiment
- Company categorization
- Labeling parts of speech
- <http://github.com/semiosis/prompts/blob/master/prompts/tweet-sentiment-classifier.prompt>
- <http://github.com/semiosis/prompts/blob/master/prompts/keyword-extraction.prompt>

Generation

■ Idea Generator

Come up with silly inventions.

The screenshot shows the OpenAI Playground interface. The main content area displays a list of 13 silly inventions generated by GPT-3. The interface includes a top navigation bar with 'OpenAI', 'Documentation', 'Playground', 'Examples', and 'Resources'. On the right side, there are settings for 'Engine' (set to 'davinci'), 'Response Length' (set to 'max'), 'Temperature' (set to '0.7'), 'Top P' (set to '1'), 'Frequency Penalty' (set to '0'), 'Presence Penalty' (set to '0'), 'Best Of' (set to '1'), 'Stop Sequences' (set to 'The end of the world'), 'Instruct Start Text' (set to 'I want to start with'), 'Instruct End Text' (set to 'I want to end with'), and 'Show Probabilities' (set to 'Off'). The list of inventions is as follows:

- 1. CuiCat**
Released at the height of the tech boom in the late 1990s, the CuiCat was a massively expensive failure. Millions of the cat-shaped bar-code scanners were produced and shipped for free across the U.S., in hopes that people would use them to scan specially marked bar codes to visit Internet sites. (How this was easier than a typing a link, the company never did answer.) Despite a much ballyhooed launch, with CuiCat codes printed in Wired and BusinessWeek, consumers never got into the idea of reading their magazines next to a wired cat-shaped scanner, and the CuiCat became little more than a high-tech paperweight.
- 2. Baby Caper**
In the 1970s, London residents lacking space for their young ones resorted to the Baby Caper. It's exactly what it sounds like: a creepy wire contraption, patented in the U.S. in 1932, that lets you claim that space outside your city window for your infant. Risky? Maybe, but so convenient. Now, sweat, how risk.
- 3. Pet Spa**
It's a washer and dryer — for your pet. Covered hilariously by CBS News in 2006, Pet Spa was a hands-free way to wash your cat on dog. Luckily for them, most people are still doing it the old-fashioned way.
- 4. The Baby-on-Board Sign**
As a proud new father, I'm a big supporter of this baby-on-board sign. But as for the rest of the world, I say this: You're either a grown-up, or you're a baby. And a sign just isn't necessary.
- 5. The Clapper**
The Clapper, which turns lights on and off in sync with the sound of your voice, was another much-ballyhooed tech bust. It seems like a great idea, but a few folks at home drew that it just doesn't work that well.
- 6. The Cuckoo Clock**
Cuckoo clocks are cool, but they're also annoying. And the cuckoo bird, with its one-note song, is downright irritating. So when you come right down to it, I'm grateful for the invention of the cuckoo clock's little cousin, the digital duck.
- 7. The Wireless Fan**
The wireless fan is the fan of the future, or so its makers would have us believe. In reality, it's the fan of the 1990s.
- 8. The Egg Carton**
While it's true that the egg carton is a great way to store eggs, I'm not so sure it's the greatest way to get them out.
- 9. The Pet Rock**
The Pet Rock was a fail that ran its course, but it's easy to see why it was a fail. It was made of stone. It was cheap. It was a pet. It was a rock. The Pet Rock was a rock for Pet's sake. (We don't need a nender description than that.)
- 10. The Obligatory But Awesome Duck-on-the-Head Shot**
This is an example of a really wonderful idea that just doesn't work. It's also a great example of a company mistaking that it does work. And the reason that it doesn't work — a duck on your head — is also a pretty good reason to keep it around.
- 11. The Muesli**
Invented in the '50s, the Muesli was a brilliant way to heat food — by exploding it. No, seriously. It was a microwave with a twist. First, you put the food in a bowl and put it in the microwave. Then you hit the button with your hand, which sent a jolt of electricity to the food. The electricity then exploded the food, killing the bacteria and heating the food all at the same time. Unfortunately, the Muesli was also a bit hazardous and it was never sold in the United States.
- 12. The Trigonator**
The Trigonator is a great idea — if you don't mind looking like a duck.

At the bottom of the list, there is a 'Submit' button and a 'Share' icon. The bottom right corner of the interface shows a '100' character count and a 'Submit' button.

Conversation

- Q&A agent
- Sarcastic chatbot

`http://github.com/semiosis/prompts/blob/master/prompts/sarcastic-response.prompt`

Design patterns

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

1. Reflective description of the task

State what the prompt does at the start At the start of the example we state in plain language what the classifier does:

```
1 _This is a tweet sentiment classifier._
```

By stating this up front
it helps the API understand much more quickly what the goal of the response is supposed to be
and you'll end needing to provide fewer examples.

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

2. Use separators between examples

Example: ###.

You can use other characters or line breaks, but ### works pretty consistently and is also an easy to use stop sequence.

Whatever separator you use, make sure that it's clear to the API where an example starts and stops.

Design patterns

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

Improving the classifier's efficiency Part 1

Make a prompt more efficient / cheaper.

Design it to generate multiple results from one API call.

```
1 This is a tweet sentiment classifier
2 Tweet: "I loved the new Batman movie!"
3 Sentiment: Positive
4 ###
5 Tweet: "I hate it when my phone battery dies"
6 Sentiment: Negative
7 ###
8 Tweet: "My day has been "
9 Sentiment: Positive
10 ###
11 Tweet: "This is the link to the article"
12 Sentiment: Neutral
13 ###
14 Tweet text
```

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

Improving the classifier's efficiency Part 2

```
1 1. "I loved the new Batman movie!"
2 2. "I hate it when my phone battery dies"
3 3. "My day has been "
4 4. "This is the link to the article"
5 5. "This new music video blew my mind"
6
7 Tweet sentiment ratings:
8 1: Positive
9 2: Negative
10 3: Positive
11 4: Neutral
12 5: Positive
13
14 ###
15 Tweet text
```

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

Improving the classifier's efficiency Part 3

```
1  "I can't stand homework"
2  "This sucks. I'm bored "
3  "I can't wait for Halloween!!!"
4  "My cat is adorable "
5  "I hate chocolate"
6  Tweet sentiment ratings:
7  1.
```

Query Reformulation

<https://www.sciencedirect.com/topics/computer-science/query-reformulation>

You can improve the quality of the responses by making a longer more diverse list in your prompt.

One way to do that is to start off with one example, let the API generate more and select the ones that you like best and add them to the list.

A few more high-quality variations can dramatically improve the quality of the responses.

pen.el: Prompt Engineering in emacs

pen.el facilitates the creation, development, discovery and usage of prompts to a Language Model such as GPT-3.

- Create elisp functions based on GPT-3 prompts
- Chain GPT-3 queries together using keyboard macros and functions
- Interactively query, generate and transform both prose and code
- Use GPT-3 as a search engine within emacs

pen.el modes Part 1

Prompt-Engineering Minor Mode

`prompt-engineering-mode` is a global minor mode for emacs that provides keybindings for creating and executing prompts generally across emacs.

Prompt Description Major Mode

`prompt-description-mode` is a major mode for editing `.prompt` files.

The `.prompt` file format is based on YAML and an associated schema, which defines the keys which are expected.

pen.el modes Part 2

Pen Messenger Minor Mode

`pen-messenger-mode` is a minor mode for enhancing an emacs-based messenger client with GPT-3 capabilities, such as emoji generation.

Pen Conversation Mode `prompt-conversation-mode` is a major mode designed to facilitate ongoing conversation with a prompt-based GPT-3 chatbot.

GPT-3 Training Mode The goal of this mode is to facilitate the workflow of training on OpenAI's API.

Prompt YAML format Part 1

meeting-bullets-to-summary.prompt

```
1 title: "meeting bullet points to summary"
2 prompt: |+
3     Convert my short hand into a first-hand
4     account of the meeting:
5
6     <1>
7
8     Summary:
9 engine: "davinci-instruct-beta"
10 temperature: 0.7
11 max-tokens: 60
```

Prompt YAML format Part 2

meeting-bullets-to-summary.prompt

```
1 top-p: 1
2 frequency-penalty: 0.0
3 presence-penalty: 0.0
4 best-of: 1
5 stop-sequences:
6 - "\n\n"
7 conversation-mode: no
8 stitch-max: 0
```

stitch-max Keep stitching together until reaching this limit. This allows a full response for answers which may need $n \times \text{max-tokens}$ to reach the stop-sequence.

meeting-bullets-to-summary.prompt

```
1  vars:
2  - "notes"
3  examples:
4  - |+
5      Tom: Profits up 50%
6      Jane: New servers are online
7      Kjell: Need more time to fix software
8      Jane: Happy to help
9      Parkman: Beta testing almost done
```

Prompts as functions

```
pen-generate-prompt-functions
```

Generate prompt functions for the files in the prompts directory
Function names are prefixed with pen-pf- for easy searching.
<http://github.com/semiosis/prompts>

exemplary: examples as functions

An example-oriented DSL that can be used to construct and compose NLP tasks.

Why is a DSL needed for this? Just to make the code a little more terse.

Regex

<https://github.com/pemistahl/grex>

```
1 (def regex
2   "example 1\nexample2" "^example [12]$"
3   "example 2\nexample3" "^example [23]$"
4   "pi4\npi5" "^pi[45]$")
```

exemplary: examples as functions

Analogy

```
1 (def analogy
2   ;; Each line is a training example.
3   "NNs" "NNs are like genetic algorithms in
4   that both are systems that learn from
5   experience"
6   "Social media" "Social media is like a
7   market in that both are systems that
8   coordinate the actions of many
9   individuals.")
10
11 (def field
12   "chemistry" "study of chemicals"
13   "biology" "study of living things")
```

Something funny

Vexate a simple instruction

```
source › git › mullikine › prompt-engineer-mode › prompts › obfuscate-language.prompt
1 Prompt: |+
2 The human asks the AI to make a simple description more complicated. The AI responds by
  writing a verbose response full of jargon to explain a simple concept.
3
4 Human: explain how to use a drinking fountain.
5
6 AI: To activate the mechanized dihydrogen monoxide dispensary service station, use the flanges
  to provide a downward force to the simple circular activation console and continue to apply
  force until the pneumatic system begins operating as intended, which can be surmised at a time
  when the dihydrogen monoxide travels in a parabolic arc which at is greatest exceeds three
  inches from the rounded excess liquid receptacle. Once hydration is completed, cease applying
  force to the console and regard whether or not the dispensary has ceased expelling contents as
  expected.
7
8 Human: explain how to play with a dog
9
10 AI: Being a typical domesticated animal, having been genetically selected for controllability,
  trainability, and general anthropomorphic features and functions, the typical dog possesses
  highly specialized and developed feeding behavior. The typical feeding behavior involves routine
  sustenance intake which can be observed by monitoring changes in animal size. Routine sustenance
  intake involves active and passive feeding habits, and one can gauge animal health by observing
  the relative levels of hydration. During training, reward conditioned reinforcement is applied.
  This involves the practice of rewarding targeted behaviors through the application of
  reinforcers, with the provision that the targeted behavior is observed. Relevant to the
  discussion at hand is the typical active and passive behavior exhibited by a typical dog.
11
12 Human: explain how to <1>
13
14 AI:
15 engine: "davinci"
16 temperature: 0.8
17 max-tokens: 100
18 top-p: 1
19 frequency-penalty: 0.2
20 presence-penalty: 0.4
21 best-of: 1
22 stop-sequences:
23   - "\n"
24 inject-start-text: yes
```


Something funny

How to crack an egg

how to crack an egg

Cracking an egg is a basic procedure which can be achieved by applying a downward force to the egg. Finding the correct force may require trial and error, but is essential to the procedure. A broken egg is an indication that the force may have been too high or too low. If it is too high, the shell will be broken, but the yolk intact. If it is too low, the force will not be sufficient to rupture the membrane of the shell, requiring that additional force be applied

Ask the audience

- What type of text to generate
 - Could be code, prose, etc.

Ruby

```
https://www.twilio.com/blog/  
generating-cooking-recipes-openai-gpt3-ruby
```