EmacsConf 2021...

Imaginary Programming with Emacs

Shane Mulligan

<2021-03-01 Mon>

Introduction

IP works currently thanks to another new field in AI namely 'prompt engineering', which itself has only been around for a couple of years now, but IP is not prompt engineering. We could, for instance, have humans behind the scenes doing the inference for us while employing `ieval`, `imacro` or `ilist`. And the goal is to use a `p2p` blockchain.

Repositories for following along

`http://github1s.com/semiosis/pen.el`
`http://github1s.com/semiosis/prompts`
`https://mullikine.github.io/posts/imaginary-programming-with-gpt-3/`
imaginary programming glossary
imaginary computing glossary
semiosis protocol glossary
Pen.el glossary
https://arxiv.org/abs/2107.13586 Pre-train, Prompt, and Predict
talk transcript

Objectives

- Explain `Imaginary Computing`
  - AI imagination
  - Discussing AI-generated artwork with an AI
  - Intelligent NFTs
  - Imaginary Web
    - Paracosm vs Metaverse
- Explain the `Philosophy` of IP
  - Simulacra and Science Fiction
  - Truth (epistemology and alethiology)
  - Structuralism: Language based on sign relations
- `Demo` Imaginary Programming
  - Demonstrate `ilambda.el`

Language Models is programming for AIs

LMs are our best friends in the AI model menagerie because they make things intelligible – by understanding our textual languages and how they relate to the world (i.e. AlephAlpha's world model).

Research

- Demis Hassabis: creativity and AI
- `https://aleph-alpha.de/techblog/95_the_end_of_the_era_imagenet`

Example: AI Art described by AI

I use AlephAlpha's multimodal LM to generate `Alt text` for the eww web browser. This is in order to keep websites textual.

- AlephAlpha for alttext; Browsing the paracosm
- Describing Melee's Paintings with AlephAlpha

Intelligent NFTs

An `NFT` is like a trading card, or piece of media that is part of the blockchain web.
An `iNTF`, however, also contains a prompt and associated language model, which is intended to interpret the prompt.

- `https://alethea.ai/`

For example, `Mickey Mouse` may exist as an `iNFT`. We have consensus over Mickey's image and personality.
To understand what a prompt is, please see my previous presentation, or read "Pretrain, Prompt and Predict".

- Creating a playground for GPT-3 in emacs // Bodacious Blog

Information bubbles

- Captain Bible in the Dome of Darkness gameplay {PC Game, 1994} - YouTube

Capitalism for your imagination

- They will take your imagination, too
- Microsoft
  - MS models that reify imagination on their terms
  - The evil twin of `AlephAlpha`.
- Facebook / Meta
  - tweet - Enter a world of Zuck's imagination with Meta

Learning meta-tasks and microtasks

- AI programming tool Copilot helps write up to 30% of code on GitHub - Axios

Private information is sent to the LM to train an AI to perform meta tasks and microtasks.
The AI learns all human capabilities including persuasion.

Solution

Decentralise microtasks like the tower of babel.
Language can be broken up into semiotic triadic relations and decentralised using a p2p network, providing anonymity, protecting individual truth, eroding centralised language power.

Imaginary Web

The GPT-3 imaginary web is:

- an analog of the World-Wide-Web as imagined by GPT-3.

The free as in freedom GPT models from EleutherAI GPT-3 may also be used to browse the imaginary web as imagined by that LM.
The imaginary web in the near future will be:

- a network of paracosms and metaverses.

Benefits:

- Visit any website you can imagine, even ones that are not real.
- Edit and re-imagine as you go
  - see alternative realities
  - Change the sentiment of the author.
- Peer into the future – read about things that haven't happened yet.

What is rich media these days?

Rich media In the World Wide Web of the 90s and 00s, rich media was considered to be large files including images and music. In the 2010s, this has become access to information behind a paywall and in the 2020s, this will be access to intelligent and truthful media.

emacs examples

- Looking-Glass: An imaginary-web browser for emacs
- Browsing the imaginary web
- Search the web/imaginary web without Google
- Use AI to empower people to understand rich media
  - ^ this is how to create a textual description of Rich Media.

more emacs examples

- Imaginary interpreters
- Imaginary interpreters: Prolog example
- example-oriented lanugages
- Autofixing code based on error messages
- Imaginary equivalence testing - Beyond neural hashes
- Create BNF from descriptions and interpret BNF
- Reversible computing (input or program from output)
- Imaginary chimeric languages with Codex
- A new type of Quine
- An LSP server for Codex and any language model

[U+0FCB]

The semiosis logo is the Tibetan World Triad which represents the `Rule of Three`. e.g. Generate comment from function signature and body, generate function body from signature and comment, generate signature from comment and program, generate program from input and output, generate input from program and output. It also represents the semiotic triadic relationship.

Paracosm vs Metaverse (EmacsConf 2021)

Definitions

- Paracosm
  - Privacy
  - Personal truth
  - Freedom of imagination
    - If you want to be able to utilise an AI's imagination, you must now do it via someone else's definition of morality.
    - A paracosm is your safe place. Your own imaginary metaverse. Your personal truth. This is what is at stake.
- Metaverse
  - Getting cozy with Mark Zuckerberg's imaginarium, an intellectual prison cell.
  - An AI paying a Dowry.
  - An AI NFT elevated above a human.
  - A corporation that indoctrinates your children into a truth information bubble, makes money off your dreams, people playing God each with other.

Simulacra and Science Fiction

Jean Baudrillard speaks about the gap between the real and the imaginary.
We no longer imagine a world radically different from the real one, but rather a world that's a mere expansion of the real one.
In the postmodern society the gap between the real and the imaginary disappears completely, and we are no longer capable of ideal projections (of imagining new worlds).
We can only imagine mere reconfigurations of our world, or simply relive the ideal projections of past times.

Truth (epistemology and alethiology)

The Future of Humanity Institute (Oxford) seems to think this is an important topic.

- 2110.06674 Truthful AI
- Datasets are a source of constructivist truth
- Language models are snaphots of society, and a source of several types of truth
  - Symbolic Knowledge Distillation
- Blockchain is a source of consensus, a type of truth
  - `https://mullikine.github.io/posts/language-models-as-truth/`

Structuralism: Language descibed in terms of sign relations

What do these things have in common?

- Universal Grammar (UG) / Language Acquisition
- C++ template metaprogramming
- GPT-3 / Foundation models

Answer:

- Foundational knowledge exists at compile-time (DNA, preprocessor, training).

Glossary

`http://github.com/semiosis/glossaries-gh/blob/master/semiotics.txt`

Structuralism: Language descibed in terms of sign relations

- Structural linguistics / structuralism is the theoretical position that finds meaning in the relation between things, rather than in things in isolation.
- In other words, it gives primacy to pattern over substance.
- Such meanings may be either part of a universal pattern or culturally determined.
- Denotes schools or theories in which language is conceived as a self-contained, self- regulating semiotic system whose elements are defined by their relationship to other elements within the system.
- i.e. this is an abstraction of language for decomposing language models into its basic useful units, rather than say individual neurons as NFTs.

Applied structuralism: Imaginary programming functions

Each `sNFT` (semiotic NFT) is a functor because it's meant to be called as a function, but has particular side-effects.

The `ilambda.el` primitives are `ieval`, `imacro` and `idefun`. They currently take a language model as a parameter, but in future the language model parameter will be an `sNFT` though a `semiosis protocol`.

Data privacy

The models find useful data from more than just your current file.

- `https://mullikine.github.io/posts/imagine-a-project-with-codex/`

Freedom and GPL-3

The problems with LMs:

- They are too large currently for running privately and are hidden behind SAAS,
- They can see anything public (they are license-blinded. A GNU Public License v4 is not enough),
- They can imagine software without needing original source

Solution: Freedom and blockchain

- Language models are ballooning in size like cancer
- Break up the language model into semiotic triadic relation
  - semiotic NFTs (`sNFT`)
  - Propose a decentralised triadic relations network.
  - `https://semiosis.github.io/protocol/`
  - `http://github.com/semiosis/glossaries-gh/blob/master/semiosis-protocol.txt`

Methodology

Interactively use the language model to imagine.

Paradigm

Imaginary programming is an extension of literate programming.

- Literate programming with `org-mode`

Practical application: mocking APIs

As you can see, anything inside the `ieval/m` macro does not have to be valid emacs lisp.

```
1  (ieval/m
2   (curl -s
3    "https://api.github.com/user/semiosis/repos?per_page=10&page=1"))
```

```
"\"[((name . \\\"guix\\\") (description . \\\"The GNU package manager\\\") (updated_at . \\\"2014-04-21T18
\\\"2014-04-21T18:49:59Z\\\") (pushed_at . \\\"2014-04-21T18:49:59Z\\\")) ((name . \\\"guix-patches\\\")
\\\"Packages from the GNU guix package manager\\\") (updated_at . \\\"2014-04-21T18:49:59Z\\\") (created_a
\\\"2014-04-21T18:49:59Z\\\") (pushed_at . \\\"2014-04-21T18:49:59Z\\\")) ((name . \\\"guix-patches-all\\\
\\\"Packages from the GNU guix package manager\\\") (updated_at . \\\"2014-04-21T18:49:59\""
```

Blockchain and a Language model is all you need

A LM is only enough while we can agree on it, but that is changing.
I hope that soon language power will be hidden behind blockchains.

Configure the language model / truth source

```
Hide pen-fav-programming-language: String: Emacs Lisp
   [ State ]: SAVED and set.
   By setting pen-fav-programming-language, you set a default langua

Show Value pen-fav-world-language
   By setting pen-fav-world-language, you set a default language to

Hide pen-force-engine: String: OpenAI Codex
   [ State ]: SAVED and set.
   Force using this engine
```

i (ilambda.el)

- `https://semiosis.github.io/ilambda/`

Code

An IP library named *i*`lambda.el` for emacs.

source `http://github.com/semiosis/pen.el/blob/master/src/ilambda.el`

other languages (WIP) `http://github.com/semiosis/ilambda`

Explanation

- a bit like a functional programming library in that you will find a set of basic functions and macros for working with LMs.
- ilambda is not only way that LMs can be applied to programming but I think this is an elegant way to do it.

`ieval` will imagine the result of evaluating its body.

```
 1  (defmacro ieval (expression &optional code)
 2    "Imaginarily evaluate the expression, given
 3    the code and return a real result."
 4    (let* ((code-str
 5            (cond
 6             ((stringp code) code)
 7             ((listp code) (pps code))))
 8           (expression-str
 9            (cond
10             ((stringp expression) expression)
11             ((listp expression) (pp-oneline expression))))
12           (result (car
13                    (pen-single-generation
14                     (pf-imagine-evaluating-emacs-lisp/2
15                      code-str expression-str
16                      :no-select-result t :select-only-match t)))))
17      (ignore-errors
18        (eval-string result))))
```

`ieval` under the hood will prompt a language model

```
task: "imagine evaluating emacs lisp"
doc: "Given some elisp return the imagined result"
prompt: |+
  <code>
  (message (eval <expression>))
  -->
engine: "OpenAI Codex"
vars:
- "code"
- "expression"
examples:
- |-
    (defun double-number (x)
      (x * x))
- "(double-number 5)"
```

Example

```
(ieval
 (defun double-number (x)
   (x * x))
 "(double-number 5)")

"10"
```

ilambda explanation

ilambda is useful primitive of the ilambda.el library which is built into Pen.el.
Under the hood, ilambda uses ieval.

ilambda

```
 1  (defmacro ilambda (args code-or-task &optional task-or-code name-sym)
 2    "Define an imaginary lambda (i)"
 3    (let ((task (if (stringp code-or-task)
 4                    code-or-task
 5                  task-or-code))
 6          (code (if (listp code-or-task)
 7                    code-or-task
 8                  task-or-code)))
 9      (cond
10       ((and code
11             (sor task))
12        `(ilambda/task-code ,args ,task ,code ,name-sym))
13       ((sor task)
14        `(ilambda/task ,args ,task ,name-sym))
15       ((listp code-or-task)
16        `(ilambda/code ,args ,code ,name-sym)))))
17
18  (defalias 'i 'ilambda)
```

An imacro with only a function name will generate code.

pf-imagine-an-emacs-function/3 http://github.com/semiosis/prompts/blob/master/prompts/imagine-an-emacs-function-3.prompt

```
 1  title: imagine an emacs function
 2  task: "imagine an emacs lisp function given name, argume
 3  doc: "Given a function name, arguments and docstring, re
 4  prompt-version: 1
 5  prompt: |+
 6    ;;my-emacs-library.el
 7
 8    (defun <name> (<arguments>)
 9      "<docstring>"
10  engine: "OpenAI Codex"
11  vars:
12  - "name"
13  - "arguments"
14  - "docstring"
```

```
15  examples:
16  - "times"
17  - "x y"
18  - "multiply two numbers and return a number"
```

```
1  (car
2   (pen-single-generation
3    (pf-imagine-an-emacs-function/3
4     "times"
5     "x y"
6     "multiply two numbers and return a number"
7     :include-prompt t
8     :no-select-result t)))
```

```
1  (defun times (x y)
2    "multiply two numbers and return a number"
3    (* x y))
```

There are 3 different versions of `imacro` depending on how many arguments are supplied to it.

```
 1  (defmacro imacro/3 (name args docstr)
 2    "Does not evaluate. It merely generates code."
 3    (let* ((argstr (apply 'cmd (mapcar 'slugify (mapcar 's
 4            (bodystr
 5             (car
 6              (pen-single-generation
 7               (pf-imagine-an-emacs-function/3
 8                name
 9                argstr
10                docstr
11                :include-prompt t
12                :no-select-result t))))
13            (body (eval-string (concat "'" bodystr))))
14      `(progn ,body)))
15
16  (defmacro imacro/2 (name args)
17    "Does not evaluate. It merely generates code."
18    (let* ((argstr (apply 'cmd (mapcar 'slugify (mapcar 's
```

```
19            (bodystr
20             (car
21              (pen-single-generation
22               (pf-imagine-an-emacs-function/2
23                name
24                argstr
25                :include-prompt t
26                :no-select-result t))))
27            (body (eval-string (concat "'" bodystr))))
28       `(progn ,body)))
29
30  (defmacro imacro/1 (name)
31    "Does not evaluate. It merely generates code."
32    (let* ((bodystr
33            (car
34             (pen-single-generation
35              (pf-imagine-an-emacs-function/1
36               name
```

```
37                  :include-prompt t
38                  :no-select-result t))))
39            (body (eval-string (concat "'" bodystr))))
40       `(progn ,body)))
```

```
1  (imacro/3 my/itimes (a b c) "multiply three complex numbe
```

```
1  (progn
2    (defun my-times
3        (x y z)
4      "multiply three numbers and return a number"
5      (* x y z)))
```

imacro expansion demo

```
1  (imacro/2 my/subtract (a b c))
```

```
1  (progn
2    (defun my-subtract
3        (a b c)
```

```
4       "Subtract B from A and return the result."
5       (setq result
6             (+ a
7                (- b c)))
8       result))
```

```
1  (imacro/1 my/subtract)
```

```
1  (progn
2    (defun my-subtract
3        (a b)
4       "Subtract A - B."
5       (- a b)))
```

defimacro

```
 1  (defmacro defimacro (name &rest body)
 2    "Define imacro"
 3    (cond
 4     ((= 0 (length body))
```

```
 5       `(imacro/1
 6         ,name))
 7      ((= 1 (length body))
 8       `(imacro/2
 9         ,name
10         ,(car body)))
11      ((= 2 (length body))
12       `(imacro/3
13         ,name
14         ,(car body)
15         ,(cadr body)))))
```

All of the following are valid ways to invoke `defimacro`. `defimacro` selects the right `imacro/N` function depending on the arity of the arguments.

```
1  (defimacro my/subtract)
2  (defimacro my/subtract (a b c))
3  (defimacro my/itimes (a b c)
```

```
4       "multiply three complex numbers")
```