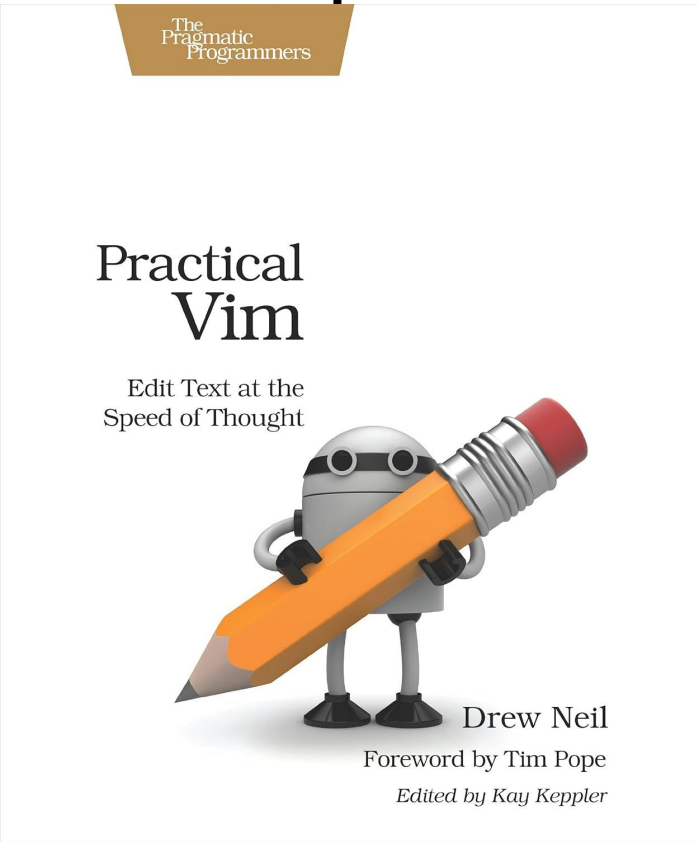
Vim meets Local LLM

Edit Text beyond the Speed of Thought

The title respects the book



About me(yuy**s13**)

- Social media accounts
  - github.com/yuys13
  - bsky.app/profile/yuys13.bsky.social
- Hobby
  - Driving on race track
- OSS
  - collama.nvim
  - asyncomplete-zsh.vim
  - pathed.fish

RP**S13**

Edit text at the speed of thought

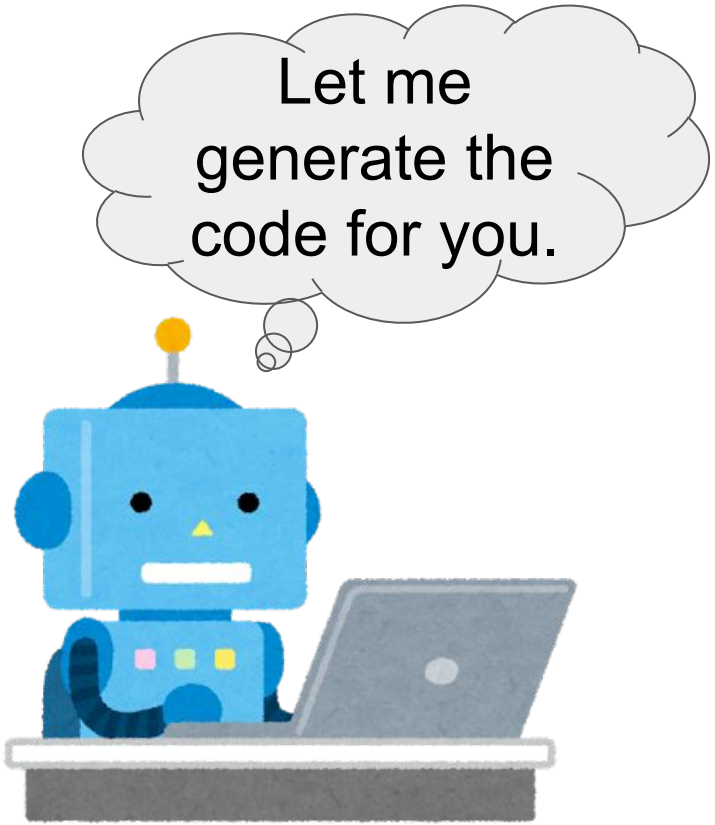
```
 1 package main
 2
 3 import "fmt"
 4
 5 // fizzBuzz prints numbers from 1 to a given number.
 6 // For multiples of 15, it prints "FizzBuzz".
 7 // For multiples of 3, it prints "Fizz".
 8 // For multiples of 5,
 9
10 func main() {
11 ¦---fizzBuzz(15)
12 }
~
~
~
~
~
~
~
~
~
~
~
~
~
main.go [+]                                    8,24          All
-- INSERT --
```

My thoughts stop

```
 1 package main
 2
 3 import "fmt"
 4
 5 // fizzBuzz prints numbers from 1 to a given number.
 6 // For multiples of 15, it prints "FizzBuzz".
 7 // For multiples of 3, it prints "Fizz".
 8 // For multiples of 5, it prints "Buzz".
 9 // Otherwise, it prints the number.
10 func
11
12 func main() {
13 ¦---fizzBuzz(15)
14 }
~
~
~
~
~
~
~
~
~
~
~
main.go [+]                                  10,6          All
-- INSERT --
```

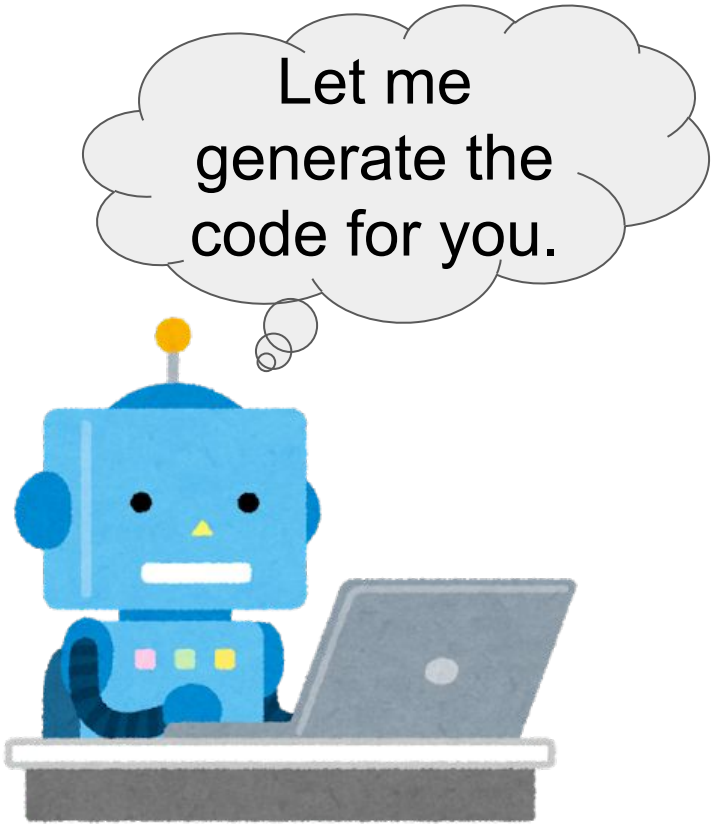
Code Generation by LLM

```
 1 package main
 2
 3 import "fmt"
 4
 5 // fizzBuzz prints numbers from 1 to a given number.
 6 // For multiples of 15, it prints "FizzBuzz".
 7 // For multiples of 3, it prints "Fizz".
 8 // For multiples of 5, it prints "Buzz".
 9 // Otherwise, it prints the number.
10 func
11
12 func main() {
13 ¦---fizzBuzz(15)
14 }
~
~
~
~
~
~
~
~
~
~
~
main.go [+]                                  10,6          All
-- INSERT --
```



Code Generation by LLM

```
 1 package main
 2
 3 import "fmt"
 4
 5 // fizzBuzz prints numbers from 1 to a given number.
 6 // For multiples of 15, it prints "FizzBuzz".
 7 // For multiples of 3, it prints "Fizz".
 8 // For multiples of 5, it prints "Buzz".
 9 // Otherwise, it prints the number.
10 func fizzBuzz(n int) {
       for i := 1; i <= n; i++ {
           if i%15 == 0 {
               fmt.Println("FizzBuzz")
           } else if i%3 == 0 {
               fmt.Println("Fizz")
           } else if i%5 == 0 {
               fmt.Println("Buzz")
           } else {
               fmt.Println(i)
           }
       }
   }
11
12 func main() {
13 ¦---fizzBuzz(15)
main.go [+]                                  10,6          All
-- INSERT --
```



Code Generation by LLM

```go
package main

import "fmt"

// fizzBuzz prints numbers from 1 to a given number.
// For multiples of 15, it prints "FizzBuzz".
// For multiples of 3, it prints "Fizz".
// For multiples of 5, it prints "Buzz".
// Otherwise, it prints the number.
func fizzBuzz(n int) {
	for i := 1; i <= n; i++ {
		if i%15 == 0 {
			fmt.Println("FizzBuzz")
		} else if i%3 == 0 {
			fmt.Println("Fizz")
		} else if i%5 == 0 {
			fmt.Println("Buzz")
		} else {
			fmt.Println(i)
		}
	}
}

func main() {
	fizzBuzz(15)
```

main.go 22,2 Top

-- INSERT --

Edit text at the speed of thought again

```
 1 package main
 2
 3 import "fmt"
 4
 5 // fizzBuzz prints numbers from 1 to a given number.
 6 // For multiples of 15, it prints "FizzBuzz".
 7 // For multiples of 3, it prints "Fizz".
 8 // For multiples of 5, it prints "Buzz".
 9 // Otherwise, it prints the number.
10 func fizzBuzz(n int) {
11 ¦---for i := 1; i <= n; i++ {
12 ¦---¦---if i%15 == 0 {
13 ¦---¦---¦---fmt.Println("FizzBuzz")
14 ¦---¦---} else if i%3 == 0 {
15 ¦---¦---¦---fmt.Println("Fizz")
16 ¦---¦---} else if i%5 == 0 {
17 ¦---¦---¦---fmt.Println("Buzz")
18 ¦---¦---} else {
19 ¦---¦---¦---fmt.Println(i)
20 ¦---¦---}
21 ¦---}
22 }
23
24 func main() {
25 ¦---fizzBuzz(15)
main.go                                    22,2          Top
-- INSERT --
```

Strong point

- Does not interfere with normal Vim usage
  - Only works when work stops during insert mode
  - Can be ignored if not needed
- Even if the generated code contains some errors, what Vim is great at is text editing, so they can work well together.

Why use a local LLM?

- Some business operations may not be able to use AI's services
  - Local LLM can be used for more business operations
- Local LLMs can be used with practical speed and accuracy with the performance of modern machines
  - The videos in this document were created using an iMac (24-inch, M1, 2021)
- It’s kind of cool, right?
- It’s fun!

Can you make a similar plugin?

Agenda

1. How to use Ollama
2. What is Fill-In-The-Middle?
3. How to create collama.nvim
4. The Future of Code Generation with LLM

1. How to use Ollama

What is Ollama?

Get up and running with large language models.

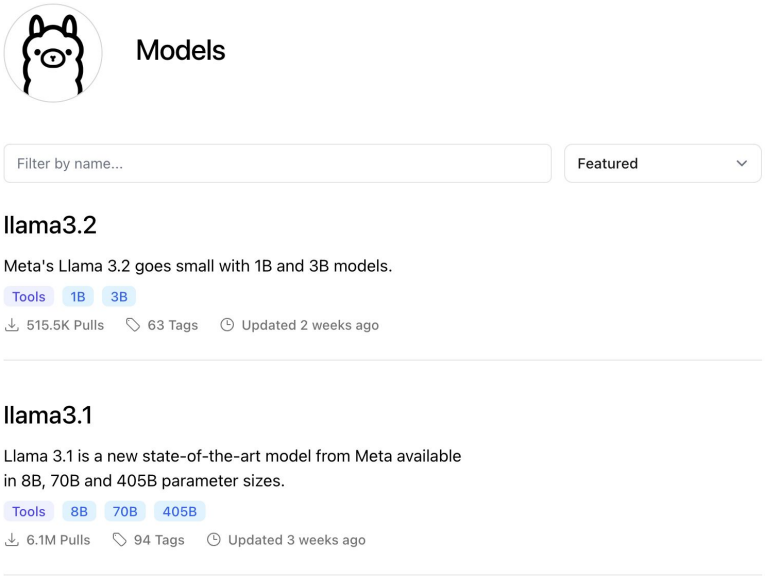
Run Llama 3.2, Phi 3, Mistral, Gemma 2, and other models. Customize and create your own.

https://ollama.com/

What is Ollama?(My Interpretation)

- LLM models can be treated like docker images
- It makes proper use of the GPU without configuration
- Separate server and CLI implementations
- Server has API via HTTP

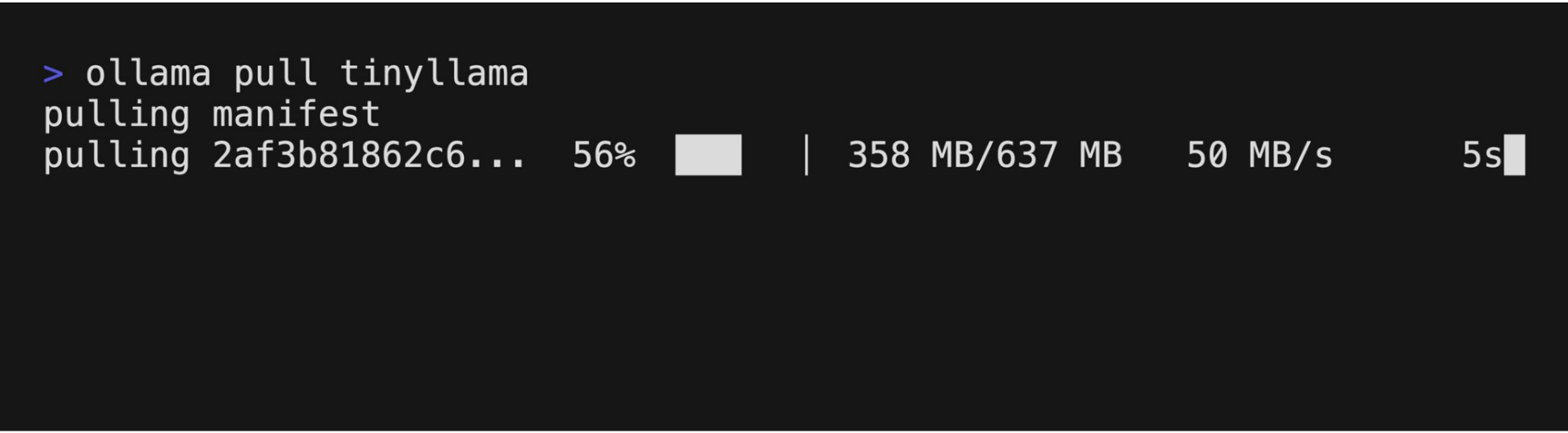
What models are available?



https://ollama.com/library

Pull a model

- ollama pull <modelname>
  - ollama pull llama3.2
  - ollama pull codellama:7b-code



Run a model

- ollama run <modelname> <prompt>
  - ollama run llama3.2 ‘Why is the sky blue?’

```
> ollama run llama3.2 'Why is the sky blue?'
The sky appears blue because of a phenomenon called Rayleigh
scattering, named after the British physicist Lord Rayleigh, who
first described it in the late 19th century.

Here's what happens:

1. When sunlight enters Earth's atmosphere
```

Let's use Ollama's API!

Generate a completion

```
POST /api/generate
```

Generate a response for a given prompt with a provided model. This is a streaming endpoint, so there will be a series of responses. The final response object will include statistics and additional data from the request.

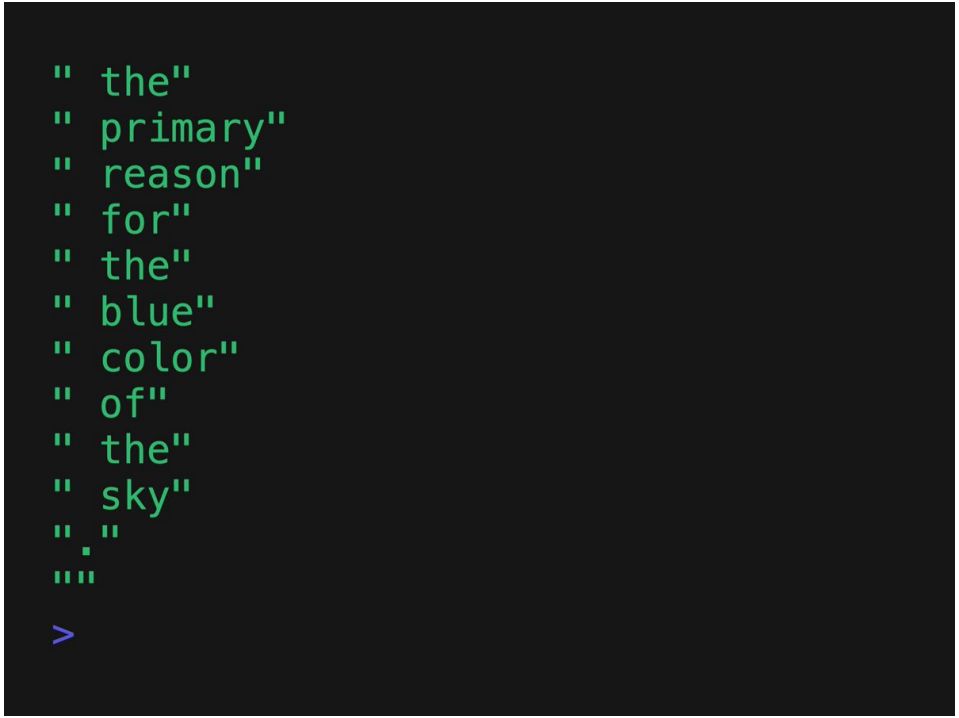
Parameters

- `model` : (required) the model name
- `prompt` : the prompt to generate a response for
- `suffix` : the text after the model response
- `images` : (optional) a list of base64-encoded images (for multimodal models such as `llava`)

https://github.com/ollama/ollama/blob/main/docs/api.md

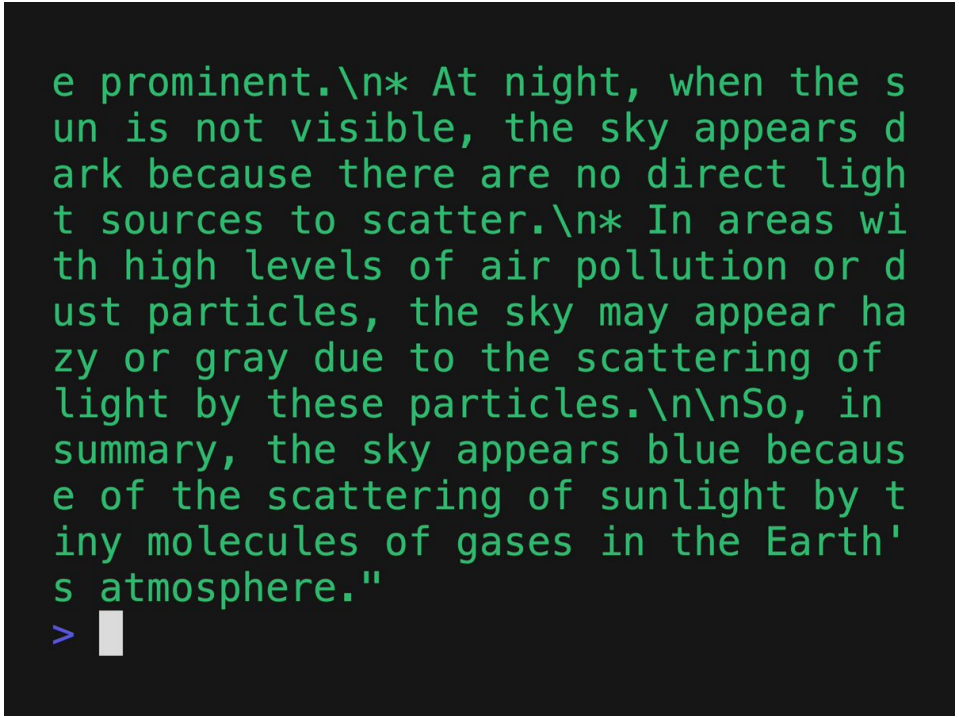
Let’s use /api/generate (stream: true)

```
curl -N
http://localhost:11434/api/generate -d
'{
  "model": "llama3.2",
  "prompt": "Why is the sky blue?"
}' | jq .response
```



Let’s use /api/generate (stream: false)

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "stream": false,
  "prompt": "Why is the sky blue?"
}' | jq .response
```



2. What is Fill-In-The-Middle?

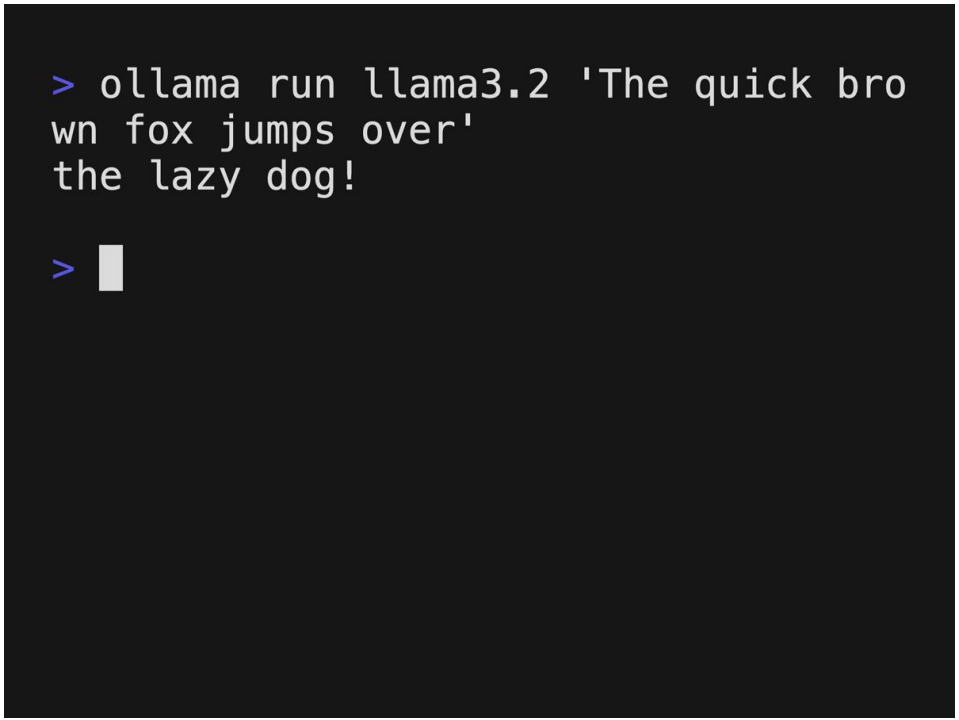
What is Fill-In-The-Middle?

https://codeium.com/blog/why-code-completion-needs-fill-in-the-middle

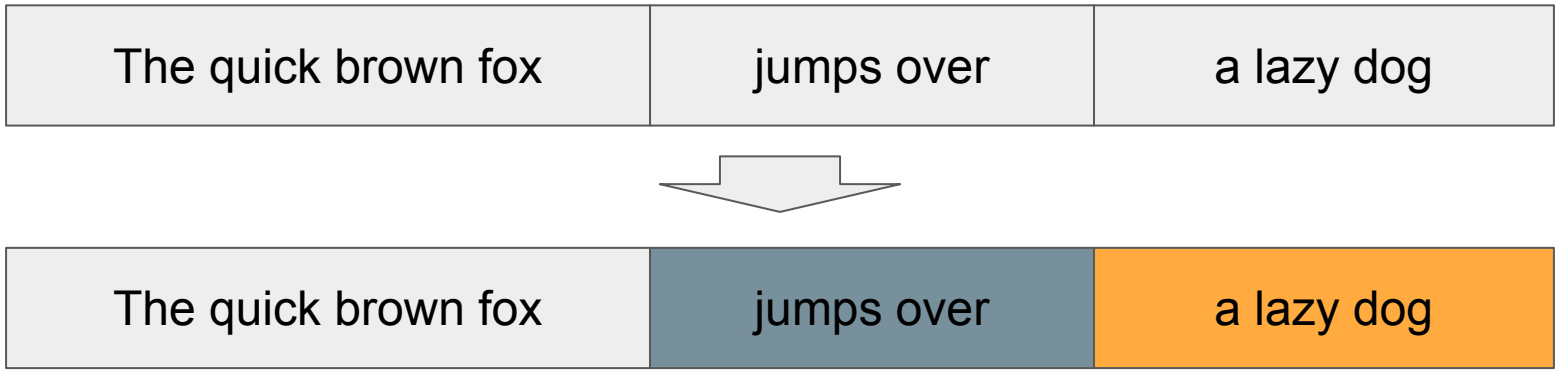
The quick brown fox jumps over a lazy dog

The quick brown fox jumps over …

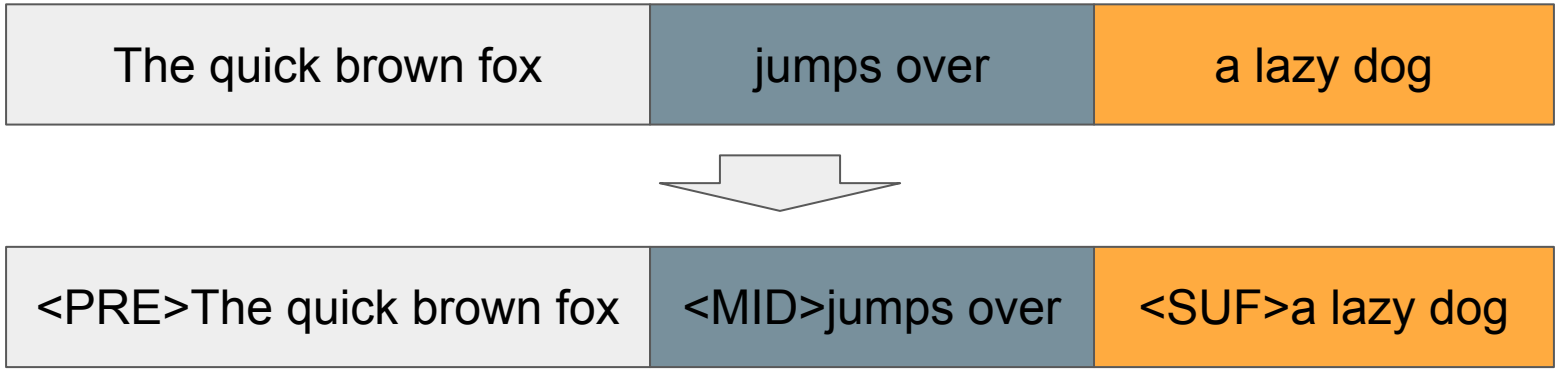
- Learn the sentence "The quick brown fox jumps over a lazy dog"
- Give the prompt "The quick brown fox jumps over"
- "the lazy dog" is generated.



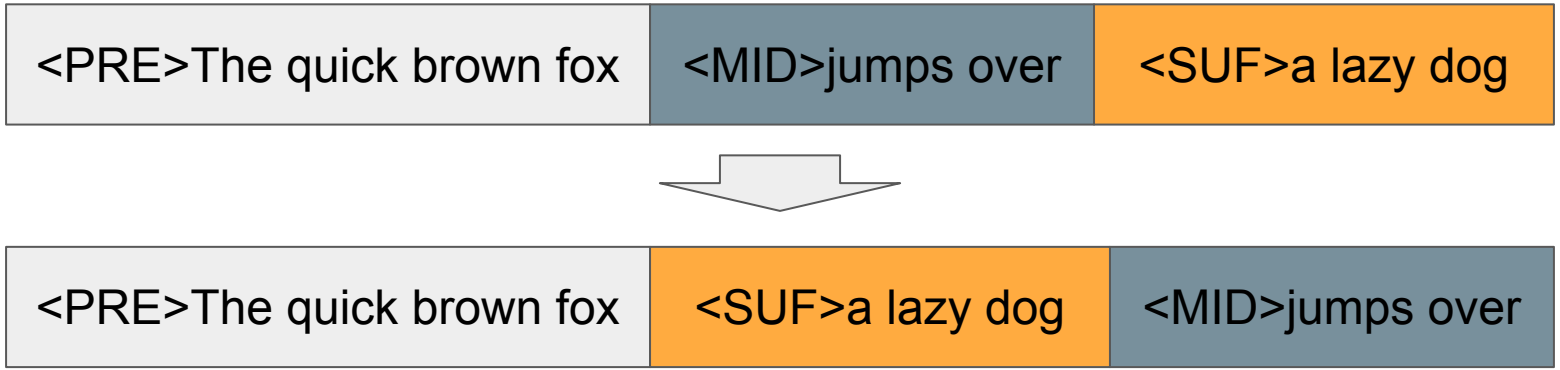
The quick brown fox … a lazy dog



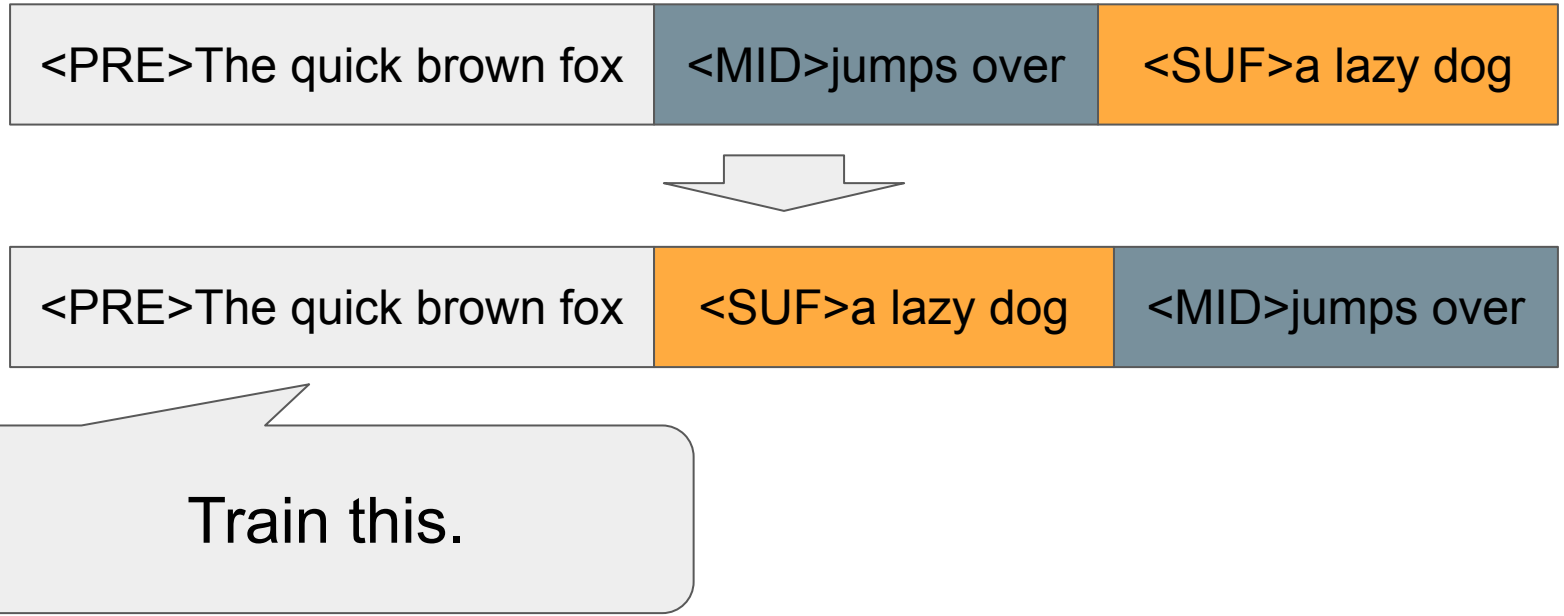
The quick brown fox … a lazy dog



The quick brown fox … a lazy dog



The quick brown fox … a lazy dog



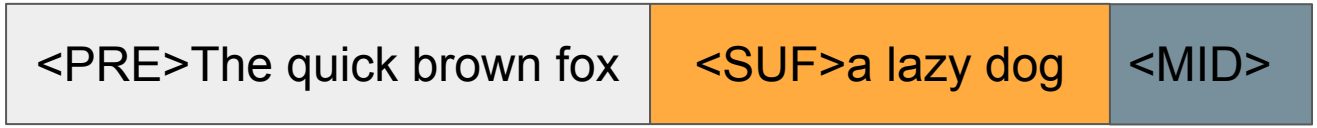
The quick brown fox … a lazy dog

Training data

The quick brown fox … a lazy dog

Training data

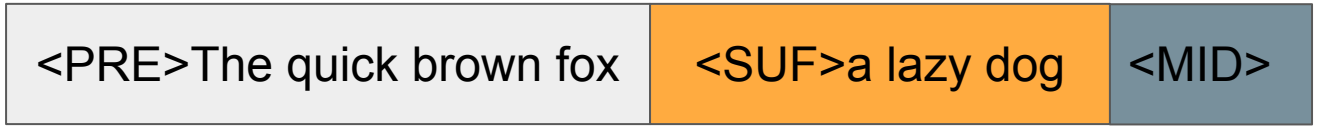
Prompt



The quick brown fox … a lazy dog

Training data

Prompt



Generated data

jumps over

Let’s try FIM

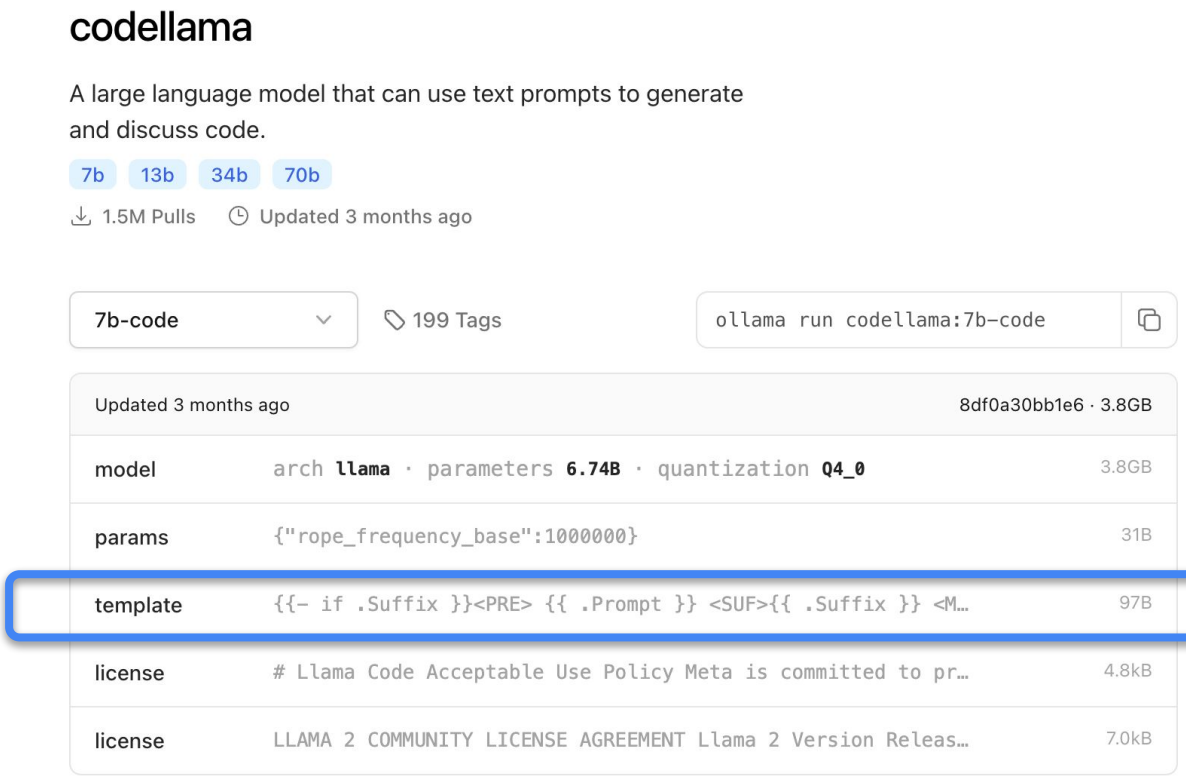
- Use codellama:7b-code
- Prefix is "The quick brown fox"
- Suffix is "a lazy dog."

ollama run codellama:7b-code '<PRE>The quick brown fox <SUF>a lazy dog. <MID>'

```
> ollama run codellama:7b-code '<PRE
>The quick brown fox <SUF>a lazy dog
. <MID>'
 jumps over

>
```

Let’s try FIM


codellama
A large language model that can use text prompts to generate and discuss code.
7b
13b
34b
70b
1.5M Pulls
Updated 3 months ago
7b-code
199 Tags
ollama run codellama:7b-code
Updated 3 months ago
8df0a30bb1e6 · 3.8GB
model
arch llama · parameters 6.74B · quantization Q4_0
3.8GB
params
{"rope_frequency_base":1000000}
31B
template
{{- if .Suffix }}<PRE> {{ .Prompt }} <SUF>{{ .Suffix }} <M…
97B
license
Llama Code Acceptable Use Policy Meta is committed to pr…
4.8kB
license
LLAMA 2 COMMUNITY LICENSE AGREEMENT Llama 2 Version Releas…
7.0kB

Let’s try FIM

codellama:7b-code / template 80d21c2229c0 · 97B

```
{{- if .Suffix }}<PRE> {{ .Prompt }} <SUF>{{ .Suffix }} <MID>
{{- else }}{{ .Prompt }}
{{- end }}
```

Let’s try FIM with API

```
curl http://localhost:11434/api/generate
-d '{
  "model": "codellama:7b-code",
  "stream": false,
  "prompt": "The quick brown fox",
  "suffix": "a lazy dog."
}'
```

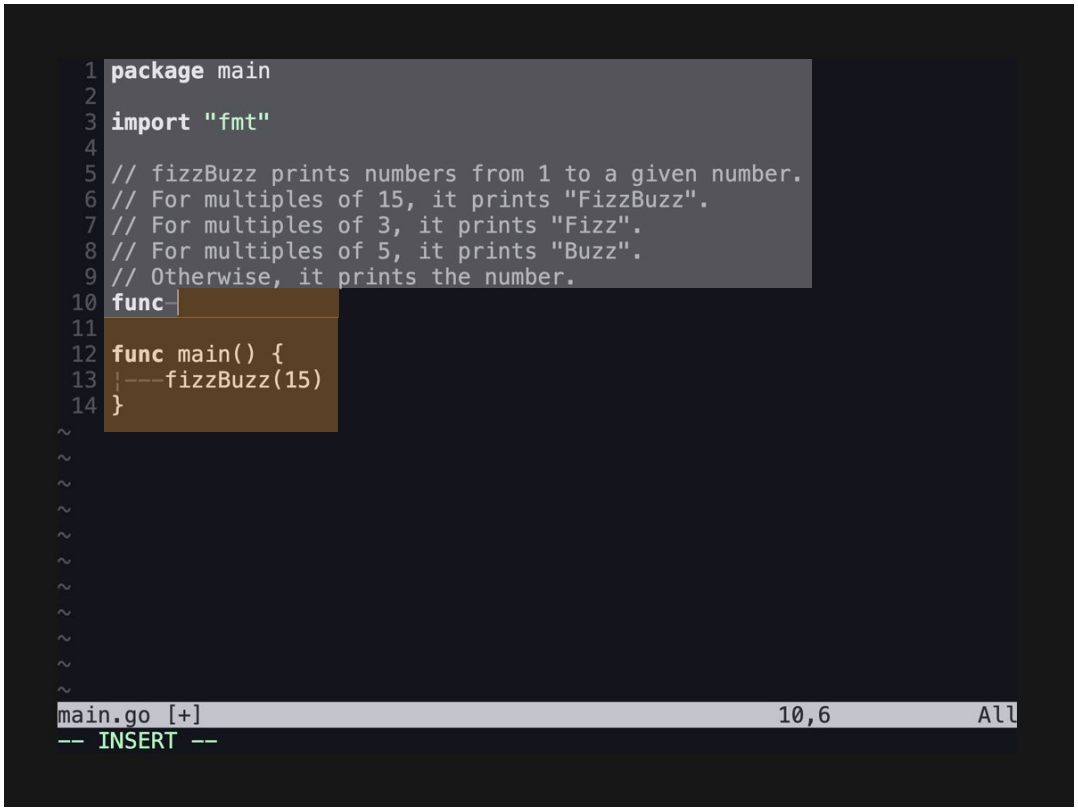
```
> curl -s http://localhost:11434/api
/generate -d '{
quote>   "model": "codellama:7b-code
",
quote>   "stream": false,
quote>   "prompt": "The quick brown
fox",
quote>   "suffix": "a lazy dog."
quote> }' | jq -r .response
 jumps over
>
```

3. How to create collama.nvim

How to create collama.nvim

1. How to get prefix and suffix
2. How to call ollama’s API asynchronously
3. How to display the generated code in a buffer
4. How to write the generated code to a buffer

How to get prefix and suffix



How to get prefix and suffix

- :h nvim_buf_get_text()
  - Gets a range from the buffer.
- Example of getting all text in the current buffer:
  - nvim_buf_get_text(0, 0, 0, -1, -1, {})
- :h nvim_win_get_cursor()
  - Gets the buffer-relative cursor position

How to get prefix and suffix

```lua
local row, col = unpack(vim.api.nvim_win_get_cursor(0))

lines = vim.api.nvim_buf_get_text(bufnr, 0, 0, row - 1, col, {})
local prefix = table.concat(lines, '\n')

lines = vim.api.nvim_buf_get_text(bufnr, row - 1, col, -1, -1, {})
local suffix = table.concat(lines, '\n')
```

How to get prefix and suffix

- :h getregion()
  - Returns the list of strings from {pos1} to {pos2} from a buffer.
  - {pos1} and {pos2} must both be |List|s with four numbers.
    - See |getpos()| for the format of the list.
- :h getpos()
- Example of getting the cursor position
  - getpos('.')
- :h line()
- :h col()
- :h searchpos()

How to get prefix and suffix

- prefix
  - getregion([0, 1, 1, 0], getpos('.'))
- suffix
  - getregion(getpos('.'), [0, line('$'), col([line('$'), '$']), 0])
  - getregion(getpos('.'), [0] + searchpos('\%$', 'n'))

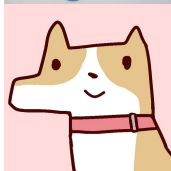
How to get prefix and suffix

- prefix
  - getregion([0, 1, 1, 0], getpos('.'))
- suffix
  - getregion(getpos('.'), [0, line('$'), col([line('$'), '$']), 0])
  - getregion(getpos('.'), [0] + searchpos('\%$', 'n'))

Thanks kuuote

Thanks thinca

Thanks utubo



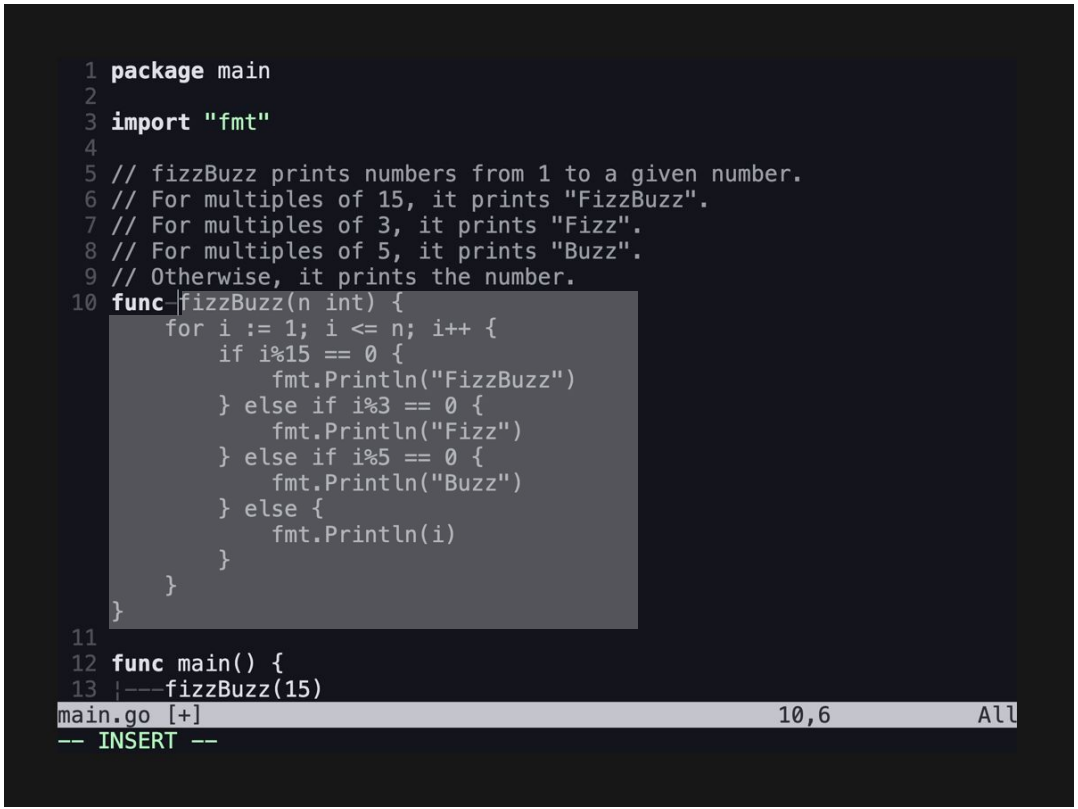
How to call ollama’s API asynchronously

- :h vim.system()
  - Runs a system command
- require('plenary.curl').post(url, opts)
  - github.com/nvim-lua/plenary.nvim
  - Curl Wrapper
- :h jobstart()
  - Note: Prefer |vim.system()| in Lua (unless using the `pty` option).
- :h uv.spawn()
  - Low level method

How to call ollama’s API asynchronously

- :h job_start()
  - Start a job and return a Job object.
  - Unlike |system()| and |:!cmd| this does not wait for the job to finish.
- github.com/ollama/ollama-js
  - with github.com/vim-denops/denops.vim
  - Denops is an ecosystem for Vim/Neovim that allows developers to write plugins in TypeScript/JavaScript powered by Deno.

How to display the generated code in a buffer



How to display the generated code in a buffer

- :h nvim_buf_set_extmark()
  - Creates or updates an |extmark|.
  - virt_text
    - virtual text to link to this mark
  - virt_lines
    - virtual lines to add next to this mark

How to display the generated code in a buffer

- :h nvim_buf_set_extmark()
  - Creates or updates an |extmark|.
  - virt_text
  - virt_lines

How to display the generated code

- :h prop_type_add()
- :h prop_add()
  - type
    - name of the text property type
  - text
    - text to be displayed
  - text_align
    - when {col} is zero; specifies where to display the text
      - below
        - in the next screen line
      - above
        - just above the line
      - etc…

How to display the generated code in a buffer

- (make-overlay BEG END &optional …)
- (overlay-put OVERLAY PROP VALUE)

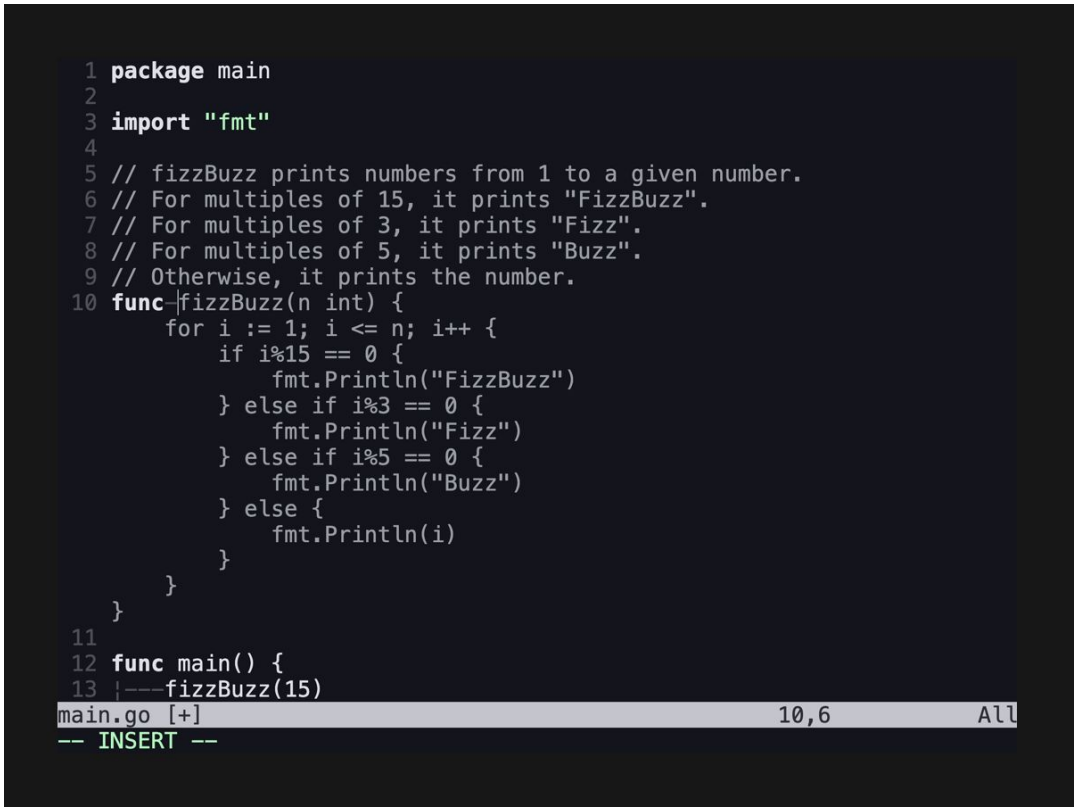
How to write the generated code to a buffer

- :h nvim_put()
  - If you want to insert text at the cursor position, this is the way to do it
  - You can also move the cursor by setting follow to true
- :h nvim_buf_set_text()
  - If you want to insert text at an arbitrary position, use this function
  - If you want to move the cursor, you need to execute nvim_win_set_cursor etc.

How to write the generated code to a buffer

- imap <expr> {lhs} collama#get_result()

Completed!


1 package main
2
3 import "fmt"
4
5 // fizzBuzz prints numbers from 1 to a given number.
6 // For multiples of 15, it prints "FizzBuzz".
7 // For multiples of 3, it prints "Fizz".
8 // For multiples of 5, it prints "Buzz".
9 // Otherwise, it prints the number.
10 func fizzBuzz(n int) {
for i := 1; i <= n; i++ {
if i%15 == 0 {
fmt.Println("FizzBuzz")
} else if i%3 == 0 {
fmt.Println("Fizz")
} else if i%5 == 0 {
fmt.Println("Buzz")
} else {
fmt.Println(i)
}
}
}
11
12 func main() {
13 ¦---fizzBuzz(15)
main.go [+] 10,6 All
-- INSERT --

4. The future of Code Generation with LLM

The future of Code Generation with LLM

- Improved quality of code generated by LLM
  - Create a PROMPT that expands the IMPORT statement
  - Give the source code in the project as context like RAG
  - Cooperate with Language Server to send relevant sources to LLM
  - etc.
- Research on UI that does not interfere with thinking
  - There must be a variety of requirements
    - I don't want the buffer to be edited, even if it is virtual text.
    - I want to check the generated code carefully with syntax highlight before applying it.
    - I want to choose from multiple candidates.
  - I limited the operation to insert mode, but is there anything that can be done in normal mode?

Conclusion

- It’s not difficult at all to use local LLM
- Code generation plugins still have room for development.
  - I want more plugins to be released
  - I would like to see more plugins that are unique and give users more choices
- Let's build a better development environment by creating and testing things together!