GitHub Copilot in RSt it's finally here!

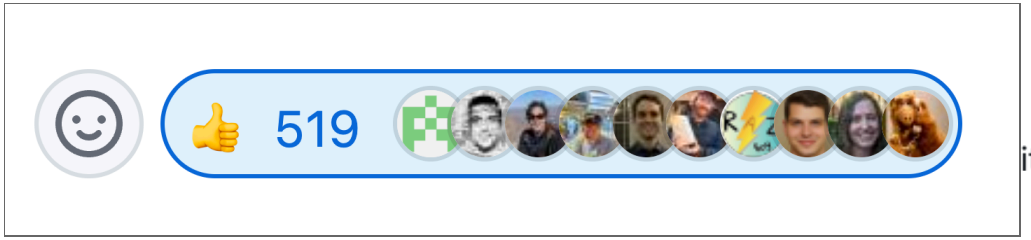
Tom Mock: Product Manager

linkedin.com/in/jthomasmock/

2024-01-18

This talk closes issue #10148

Hi!



ity that Copilot is not

I've been using Copilot with the Ruby language for the past 1-2 weeks and I'm convinced that I can't imagine programming without an aid like Copilot for the next decade.

I found several threads in the community mentioning copilot, but they were all closed.

- community.rstudio.com/t/github-copilot-and-r/108839
- community.rstudio.com/t/github-copilot-integration-with-rstudio/114967

Ok, so what's generative AI?

> Generative AI refers to a category of AI models and tools designed to create new content, such as text, images, videos, music, or code. Generative AI uses a variety of techniques—including neural networks and deep learning algorithms—to identify patterns and generate new outcomes based on them. - GitHub blog

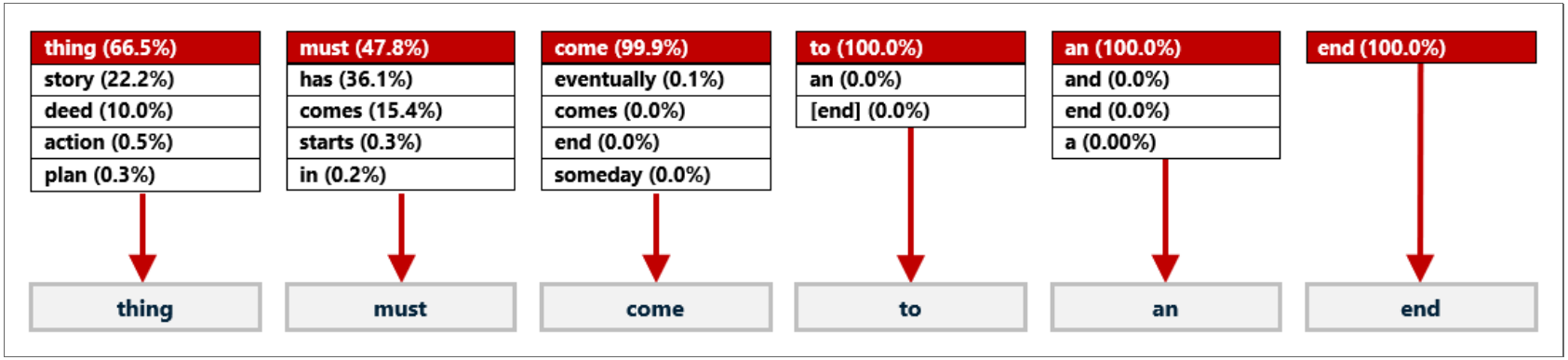
Midjourney prompt:

```
png transparent background, a seated, robotic android boston
terrier wearing pilot goggles, in the style of pixel art, full
body --style raw --stylize 50
```

Generative AI for text

For text generation, Generative AI *just* wants to predict the next word/token/string!
I might ask ChatGPT (asking is also known as “prompting”):

> “Complete the sentence `every good`.”



Source

‘Trust’ but verify

- Generative AI doesn’t have a ‘brain’ or general intelligence
- It’s just a model that’s been trained on a lot of data
- It’s not always right, appropriate, or optimal
- It can make up things that aren’t true, or use code that doesn’t actually exist (or run!)
- So it’s important to verify the output before using it
- But we can use it to quickly experiment and maybe provide a novel direction – basically “prompt” ourselves and our own knowledge

What is Copilot?

> GitHub Copilot is an AI pair programmer that offers autocomplete-style suggestions and real-time hints for the code you are writing by providing suggestions as “ghost text” based on the context of the surrounding code - **GitHub Copilot docs**

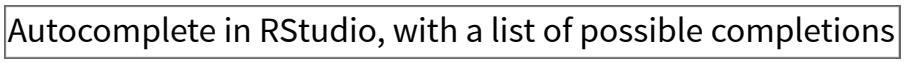
Autocomplete

- Parses code and environment
- Supplies possible completions
- Static set of completions in popup
- IDE provided from local disk

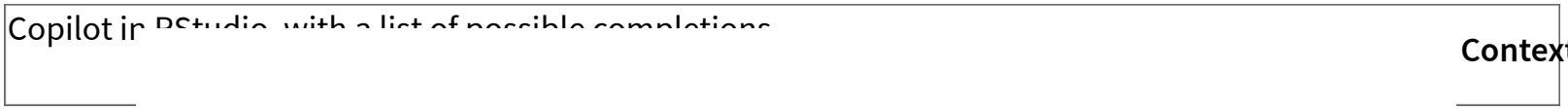
Copilot

- Parses code, environment and training data
- Supplies likely completions
- Dynamic set of completions via ‘ghost text’
- Generative AI provided via API endpoint

Autocomplete vs Copilot



RStudio, now with ‘Ghost Text’



Copilot generated ‘ghost text’

Ghost text doesn’t “exist” in document until accepted

Copilot in RStudio

Copilot in RStudio

Git	Signed-in	Sign in page

Get started

- Get a subscription to GitHub Copilot, Personal or Business
- Tools > Global Options > Copilot Tab
  - Download and install the Copilot agent
  - Sign in with your GitHub account, by using this feature you agree to the **GitHub Copilot Terms of Service**

Don’t be afraid to play around!

Keyword is a word game, popularized by the Washington Post.

How can we solve the Keyword game?
With RStudio + Copilot of course!

Getting the most out of the generative loop

GitHub Copilot is a generative AI tool that can be used to generate (ie predict) output text, and more specifically, code.

> Generative AI doesn't understand anything. It's just a prediction engine! - David Smith

- Context - What prompts have been provided?
- Intent - What is the user trying to do?
- Output - What is actually returned?

Better context == closer to intent == better output

$_{S}2_{C}$

Simple, Specific, and use Comments!

Simple and specific: Break down complex tasks

Provide a high-level description of the project goal at the top level, and build off that with more specific tasks.
For example, I know how to play Keyword, and I know others have **solved similar games (Wordle) with R**.

```
# create a function to solve the Keyword game
# this game uses a 6 letter horizontal word at
# intersection of 6 other vertical words,
# where a missing letter from each of the vert
# accounts for one letter of the mystery 6 let

# how to play
# Guess 6 letters across.
# Letters must spell words across and down.
```

Simple(st): ~~Cheat~~ Be clever

```
library(jsonlite)
url <- "https://keyword-client-prod.red.aws.wa
raw_json <- fromJSON(url)
```

```
raw_json$answer
```

```
[1] "staple"
```

Ok that's cheating, but what else is in there?

```
raw_json$words
```

```
[1] "_ee"    "chan_"  "b_re"   "s_ur"   "real_y" "scal_"

[1] "[s]ee"    "chan[t]"  "b[a]re"   "s[p]ur"
"real[l]y" "scal[e]"
```

Simple: Just get the hint words

Break it down into component parts, let’s work with the hint words!

Simple: Just get the hint words

```
raw_json <- json_url(date = "2023-08-09") |> j

hints <- as.character(raw_json$words)
hints
```

```
[1] "_ee"    "chan_"  "b_re"   "s_ur"   "real_y" "scal_"
```

Specific: Use expressive names (and comments)

- Use expressive names for variables, functions, and objects (this is best practice anyway!)

Specific: Use expressive names

top_words function with ghost-text

letters_from_blank function with ghost-text

S²C: `guess_keyword()` function

```
# Solved with Copilot and some human ingenuity
guess_keyword("2023-09-09")
```

```
The keyword is one of the following:
recipe, repipe

The played words are some of the following:
[r]aid, stat[e], [c]ap, gamb[i]t, [p]arent, decr[e]e
[r]aid, stat[e], [p]ap, gamb[i]t, [p]arent, decr[e]e
```

You can see the full "transcript" at: **https://gist.github.com/jthomasmock/a77072d61de92314149f63780e22ab21**
`guess_keyword()` function: **https://gist.github.com/jthomasmock/197d385dfefbeef61ec5ef6cce0d0ecc**

Getting stuck?

- Add more context, and follow S^2C (Simple, Specific, and use Comments)
- Prompt again or in a different way
- Add more top-level or inline comments
- Build off your own momentum (write some of your own code)
- Turn off Copilot for a bit

Better context == closer to intent == better output

More than one way to generate text

Ghost Text

Chat with `{chattr}`

{chattr}

Chat with chattr in RStudio viewer pane

{chattr} - enriched requests

```
library(chattr)

data(mtcars)
data(iris)

chattr(preview = TRUE)
```

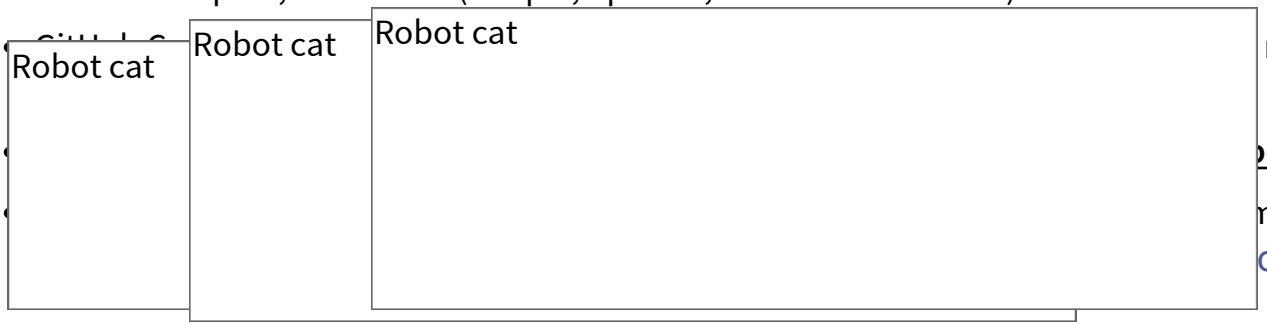
Diagram of chattr request

```
#>
#> ── Prompt:
#> role: system
#> content: You are a helpful coding assistant
#> role: user
#> content:
#> * Use the 'Tidy Modeling with R' (https://www.tmwr.org/) book as main reference
#> * Use the 'R for Data Science' (https://r4ds.had.co.nz/) book as main reference
#> * Use tidyverse packages: readr, ggplot2, dplyr, tidyr
#> * For models, use tidymodels packages: recipes, parsnip, yardstick, workflows,
#> broom
#> * Avoid explanations unless requested by user, expecting code only
#> * For any line that is not code, prefix with a: #
#> * Keep each line of explanations to no more than 80 characters
#> * DO NOT use Markdown for the code
#> [Your future prompt goes here]
```

Generative AI tools with Posit Workbench and RStudio

- For best outputs, follow S^{2}C (Simple, Specific, and use Comments)
- release of RStudio and Posit
- models such as LLaMA, and more

```
chattr")
```



Additional Slides

What about different backend models for RStudio?

There will be some options to use community-created RStudio add-ins to connect to other models and still make use of ghost text:

What other things can Copilot generate?

```
# test the time_between function with testthat
library(testthat)

test_that("time_between works", {
  expect_equal(time_between("2023-08-15", "2022-08-15", "days"), -365)
  expect_equal(time_between("2023-08-15", "2022-08-15", "weeks"), -52.1428571428571)
  expect_equal(time_between("2023-08-15", "2022-08-15", "months"), -12)
  expect_equal(time_between("2023-08-15", "2022-08-15", "years"), -1)
  expect_equal(time_between("2022-12-06", Sys.Date(), "decades"), "Please enter a va
})
```

Example of Copilot generating code

Example of Copilot generating code

`{chattr}` interface

The main way to use `chattr` is through the Shiny Gadget app. By default, it runs inside the Viewer pane, and use `as_job = TRUE` in RStudio to run it in the background:

```
# Change default to run as job via: `options(chattr.as_job=TRUE)`
chattr::chattr_app(as_job = TRUE)
```

`{chattr}`: Available models

`chattr` provides two main integration with two main LLM back-ends. Each back-end provides access to multiple LLM types:

Provider	Models	Setup Instructions
OpenAI	GPT Models accessible via the OpenAI's REST API. `chattr` provides a convenient way to interact with GPT 3.5, and DaVinci 3.	**Interact with OpenAI GPT models**
LLamaGPT-Chat	LLM models available in your computer. Including GPT-J, LLaMA, and MPT. Tested on a **GPT4ALL** model. **LLamaGPT-Chat** is a command line chat program for models written in C++.	**Interact with local models**

The idea is that as time goes by, more back-ends will be added.

What about GenAI for other IDEs in Posit Workbench?

- Posit Workbench will be upgrading JupyterLab to v4, and dev team exploring JupyterAI
- VS Code has many extensions for Generative AI, including Codeium, AWS Code Whisperer, Tabnine and more…

pos.it/rstudio-copilot by **Tom Mock**

hide-logo.js