



Using Copilot with Vim / Neovim

October 24, 2024

Rory Kelly
CISL / CSG

Background and Prereqs

This is an overview of setting up Copilot to work in Vim and Neovim. I'll briefly cover two plugins, one which works in both Vim and Neovim, and one that only works in Neovim. There are many other plugins I'm not covering, including ones for Llama, Gemini, and other models.

- [Copilot.vim](#) (official github product, works with vim/nvim)
- [CopilotChat.nvim](#) (3rd party plugin, works with nvim only)

In the remainder of this presentation, I'm assuming you're a vim user, but not necessarily an advanced one. But you should be able to open, write, and save files, navigate windows, copy/paste etc.

Copilot.vim -- Setup

Repo at: <https://github.com/github/copilot.vim>

1. Have a registered Copilot account associated with your Github account
2. Install vim (9.0.185 +) or any version of neovim.

On Derecho, using `/usr/bin/vi` or `/usr/bin/vim` will work. The vim included in the ncarenv modules is too old to work with copilot.vim.

3. Install Node.js

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash
nvm install 20
```

4. Install Copilot.vim

```
git clone https://github.com/github/copilot.vim.git \
~/.vim/pack/github/start/copilot.vim
```

5. Setup

start vim and do

```
:Copilot setup
```

Copilot.vim -- Usage

- By default Copilot will make suggestions as you type.
- You can cycle through suggestions and **accept** or **dismiss** them.
- You can also partially accept them by **word** or **line**.
- Autocomplete can also be turned off (**:Copilot disable**) and you can then ask Copilot to **suggest** a completion as desired.

```
// A C program to compute the collatz sequence of a number

#include <stdio.h>

int collatz(int n) { (2/2)
    if (n == 1) {
        return 1;
    } else if (n % 2 == 0) {
        return collatz(n/2);
    } else {
        return collatz(3*n + 1);
    }
}
~
```


Copilot.vim -- Setup

The default key bindings may or may not work well for you depending on your platform, terminal application, etc.

On a Mac with iTerm2, I did not like the default keybindings, so reassigned a few in my .vimrc (or init.lua for nvim, more on that later)

Suggestions can also be tailored a bit more specifically by letting copilot look at more of your code for context.

key bindings in ~rory/.vimrc

```
imap <silent><script><expr> <M-a> copilot#Accept("\<CR>")
let g:copilot_no_tab_map = v:true
imap <M-w> <Plug>(copilot-accept-word)
imap <M-l> <Plug>(copilot-accept-line)
imap <M-d> <Plug>(copilot-dismiss)
imap <M-s> <Plug>(copilot-suggest)
let g:copilot_workspace_folders = ["~/projects/"]
```

Back your .vimrc up before making changes!

CopilotChat.nvim -- Setup

Repo at: <https://github.com/CopilotC-Nvim/CopilotChat.nvim>

1. Have a registered Copilot account associated with your Github account
2. Install any version of neovim. Binaries available at <https://github.com/neovim/neovim/blob/master/INSTALL.md#linux>
3. Install Copilot.vim (see previous section for details)
4. Install CopilotChat.nvim with your favorite package manager
<https://github.com/CopilotC-Nvim/CopilotChat.nvim?tab=readme-ov-file#installation>
5. Setup -- nothing more to do if Copilot.vim is setup. But you can customize settings
`~/.config/nvim/init.vim` or
`~/.config/nvim/init.lua`

CopilotChat operates similarly to the 'panel' mode in Copilot.vim, but is generally more powerful, more flexible, and contains some special features.

CopilotChat commands

- `:CopilotChatOpen / :CopilotChatClose`
- `:CopilotChat <prompt>`
- `:CopilotChatModels`
- `:CopilotChatExplain`
- `:CopilotChatReview`
- `:CopilotChatFix`
- `:CopilotChatDocs`
- `:CopilotChatTests`
- `:CopilotChatModels`
- ...

CopilotChat.nvim -- Usage

:CopilotChat example

```
1 ## User
2
3 write a C function that prints the collatz sequence of a given
  input integer
4
5 ## Copilot
6
7
8 #include <stdio.h>
9
10 void print_collatz_sequence(int n) {
11     if (n <= 0) {
12         printf("Input must be a positive integer.\n");
13         return;
14     }
15
16     while (n != 1) {
17         printf("%d ", n);
18         if (n % 2 == 0) {
19             n = n / 2;
20         } else {
21             n = 3 * n + 1;
22         }
23     }
24     printf("1\n");
25 }
26
27 int main() {
28     int number;
29     printf("Enter a positive integer: ");
30     scanf("%d", &number);
31     print_collatz_sequence(number);
32     return 0;
33 }
34
```

```
1 // collatz.c
2 // This program will take a number from the user
3 // and print the collatz sequence for that number
4 //
5
6 #include <stdio.h>
7
```

CopilotChat.nvim -- Usage

:CopilotChat example

```
27 int main() {
28     int number;
29     printf("Enter a positive integer: ");
30     scanf("%d", &number);
31     print_collatz_sequence(number);
32     return 0;
33 }
34
35
36 ## User -----
37
38 that's great, but now give me the same function in python
39
40 ## Copilot -----
41
42
43 def print_collatz_sequence(n):
44     if n <= 0:
45         print("Input must be a positive integer.")
46         return
47
48     while n != 1:
49         print(n, end=' ')
50         if n % 2 == 0:
51             n = n // 2
52         else:
53             n = 3 * n + 1
54     print(1)
55
56 if __name__ == "__main__":
57     number = int(input("Enter a positive integer: "))
58     print_collatz_sequence(number)
59
```

```
1 // collatz.c
2 // This program will take a number from the user
3 // and print the collatz sequence for that number
4 //
5
6 #include <stdio.h>
```

CopilotChat.nvim -- Usage

:CopilotChatExplain example

```
1 ## User
2
3 Write an explanation for the active selection as paragraphs of
  text.
4
5 ## Copilot
6
7 The provided code defines a function named blaarg in C++. This
  function takes two parameters: an integer n and a floating-point
  number x. The purpose of the function is to compare the integer
  n (converted to a float) with the floating-point number x and
  return a value based on this comparison.
8
9 The function begins by checking if the float representation of n
  is greater than x. This is done using the expression float(n) >
  x. If this condition is true, the function returns the float
  representation of n. This conversion ensures that the comparison
  is made between two floating-point numbers, which avoids potential
  issues with comparing different data types.
10
11 If the condition float(n) > x is not met, the function proceeds
  to the else block. Here, it returns the product of x and the
  constant 1.618033, which is an approximation of the golden ratio
  ( $\phi$ ). This constant is often used in mathematical contexts and can
  be significant depending on the application's domain.
12
13 Overall, the blaarg function is a simple conditional function
  that either returns the float representation of an integer or a
  scaled version of a floating-point number based on a comparison.
  This function could be useful in scenarios where such conditional
  logic is needed, although the specific use case is not clear from
  the provided code alone.
14
```

```
1
2 // definition of the blaarg funtion
3 float blaarg(int n, float x){
4     if (float(n) > x){
5         return float(n);
6     }
7     else{
8         return 1.618033 * x;
9     }
10 }
```

CopilotChat.nvim -- Usage

:CopilotChatDocs example

```
15 ## User -----
16
17 Please add documentation comment for the selection.
18
19 ## Copilot -----
20
21
22 /**
23  * @brief Compares an integer and a float, returning a value based
24  * on the comparison.
25  *
26  * This function takes an integer `n` and a floating-point number
27  * `x`. It converts `n` to a float
28  * and compares it with `x`. If the float representation of `n` is
29  * greater than `x`, the function
30  * returns the float representation of `n`. Otherwise, it returns
31  * the product of `x` and the
32  * golden ratio (approximately 1.618033).
33  *
34  * @param n The integer to be compared.
35  * @param x The floating-point number to be compared.
36  * @return The float representation of `n` if `float(n) > x`,
37  * otherwise `1.618033 * x`.
38  */
39 float blaarg(int n, float x){
40     if (float(n) > x){
41         return float(n);
42     }
43     else{
44         return 1.618033 * x;
45     }
46 }
```

```
1
2 // definition of the blaarg funtion
3 float blaarg(int n, float x){
4     if (float(n) > x){
5         return float(n);
6     }
7     else{
8         return 1.618033 * x;
9     }
10 }
```

One feature of CopilotChat.nvim that I really like is the ability to easily switch LLM models (within the family)

:CopilotChatModels

```
Select a model
1: gpt-3.5-turbo-0613
2: gpt-4-0613
3: gpt-4o-2024-05-13
4: gpt-4o-2024-08-06
5: gpt-4o-mini-2024-07-18
6: gpt-4-0125-preview
Type number and <Enter> or click with the mouse (q or empty cancels):
```

That's a brief overview of using Copilot features in Vim and Neovim. If you are an Emacs person it works there too.

Ben will explain setting up in Emacs next.

I'm happy to take questions now or later