

AI in Xcode: is it better than the competitors?

Vincent Pradeilles ([@v_pradeilles](#)) –  **Photroom**

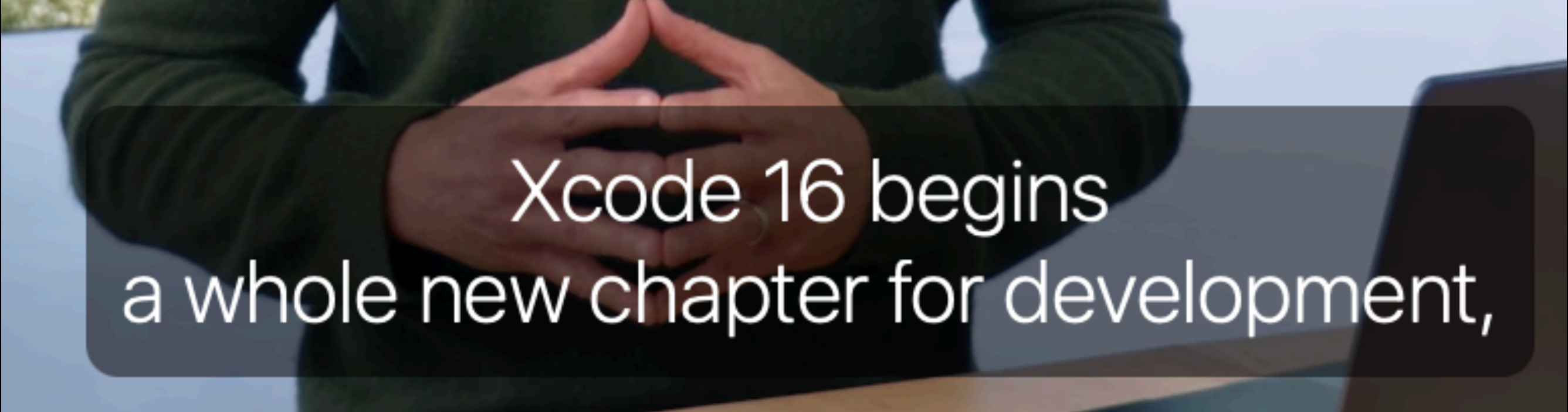
I don't know if you've heard...

...but Apple made some
AI announcements at WWDC!

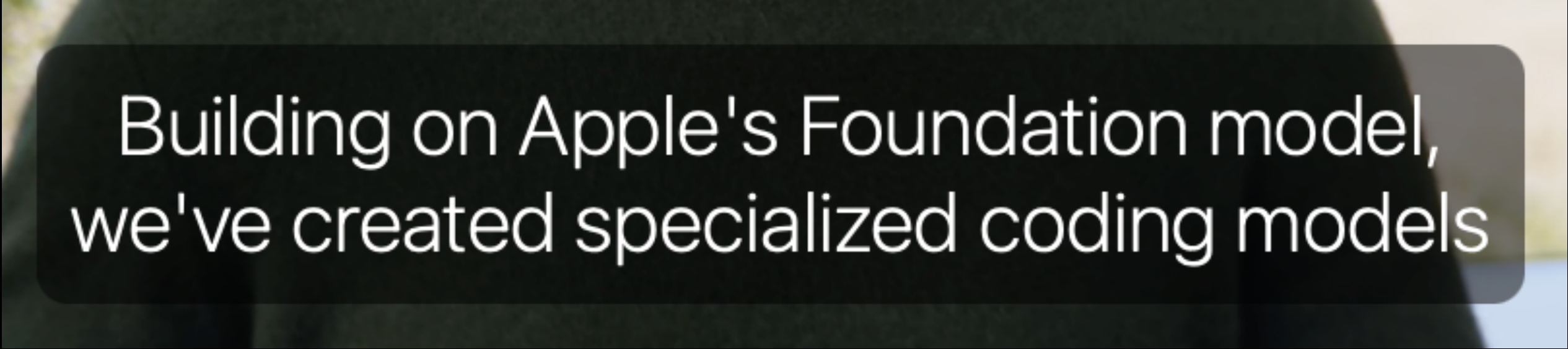
**And one announcement was
just for us, developers!**



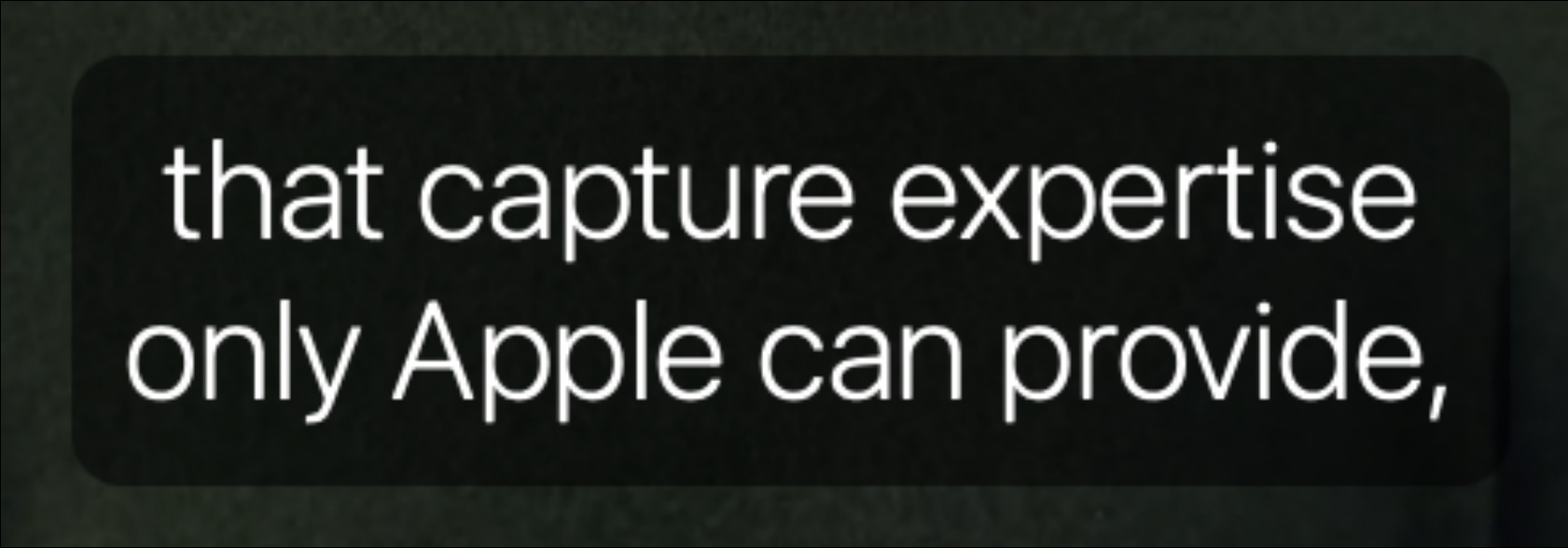
Xcode 16 begins
a whole new chapter for development,



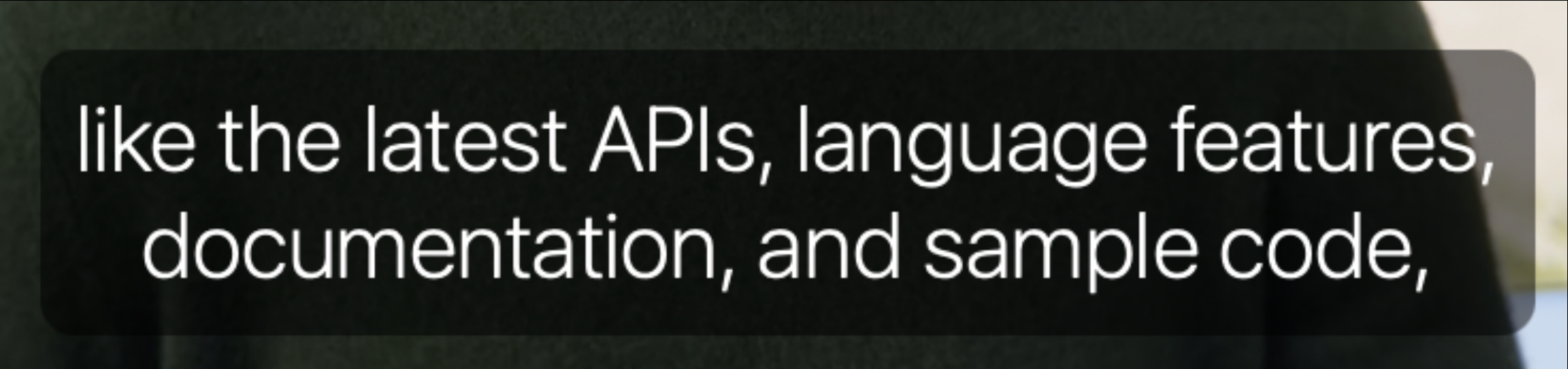
Xcode 16 begins
a whole new chapter for development,



Building on Apple's Foundation model,
we've created specialized coding models



that capture expertise
only Apple can provide,



like the latest APIs, language features,
documentation, and sample code,

But how good is it really,
compared to competitors?



Make this function asynchronous|

**Unfortunately, Swift Assist is
still unavailable for now...**


```
3 struct VideoCollectionView: View {  
4  
5     let name: String  
6  
7 }  
8
```

...so we'll focus on
Predictive Code Completion



GitHub
Copilot

Predictive Code Completion
has a very obvious competitor



Visual Studio Code



Swift v1.11.0

Swift Server Work Group

Swift Language Support for Visual Studio Code.

Disable



Uninstall



Auto Update



DETAILS

FEATURES

CHANGELOG

DEPENDENCIES

Swift for Visual Studio Code

And you can totally use Copilot
with Swift code!

It's worth noting that these two tools have opposite approaches:

- Predictive Code Completion runs locally
- Copilot runs in the cloud

So let's see how they compare
on 5 typical use cases

#01 - Implementing a data model

#01 - Implementing a data model

```
8  import Foundation
9
10 struct Address {
11     var street: String
12     var city: String
13     var zipCode: String
14     var country: String
15 }
```

#01 - Implementing a data model

```
8  import Foundation
9
10 struct Address {
11 }
12
```

#01 - Implementing a data model

Similar predictions, but Xcode did take more time to get to the full result.

(but did suggest to use `let`, which is arguably more appropriate)

#02 - Implementing a mock

#02 - Implementing a mock

```
8  import Foundation
9
10 struct Address {
11     var street: String
12     var city: String
13     var zipCode: String
14     var country: String
15 }
16
17 extension Address {
18     static var mock: Address {
19         Address(street: "1234 Main St", city: "San Francisco", zipCode: "94111", country: "US")
20     }
21 }
```

#02 - Implementing a mock

```
8 import Foundation
9
10 struct Address {
11     let street: String
12     let city: String
13     let zip: String
14     let country: String
15 }
16
17 extension Address {
18     static var mock: Address {
19         .init(street: "123 Main St", city: "Anytown", zip: "12345", country: "US")
20     }
21 }
22
```

⦿ 'static var' declaration requires an initializer expression or an explicitly stated getter

✕ Missing return in accessor expected to return 'Address'

#02 - Implementing a mock

Similar predictions and time to make them.

#03 - Implementing an array of mocks

#03 - Implementing an array of mocks

```
26 static var mocks: [Address] {  
27     [  
        Address(  
            street: "1234 Main St",  
            city: "San Francisco",  
            zipCode: "94111",  
            country: "USA"),  
        Address(  
            street: "5678 Main St",  
            city: "San Francisco",  
            zipCode: "94111",  
            country: "USA"),  
        Address(  
            street: "9101 Main St",  
            city: "San Francisco",  
            zipCode: "94111",  
            country: "USA")  
    ]  
28 }  
29 }
```


#03 - Implementing an array of mocks

```
17 extension Address {  
18     static var mock: Address {  
19         .init(  
20             street: "123 Main St",  
21             city: "Anytown",  
22             zip: "12345",  
23             country: "US"  
24         )  
25     }  
26  
27     static var mocks: [Address] {  
28         [mock  
29     }  
30 }  
31
```

#03 - Implementing an array of mocks


```
27     static var mocks: [Address] {  
28         [  
29             mock,  
30             .init(street: "456 Main St",  
                    city: "Anytown",  
                    zip: "12345",  
                    country: "US"  
                ),  
                .init(  
                    street: "789 Main St",  
                    city: "Anytown",  
                    zip: "12345",  
31             }  
32     }  
33
```

#03 - Implementing an array of mocks

You eventually get similar predictions,
but it's more laborious in Xcode

#04 - Implementing an API call

#04 - Implementing an API call

```
17 struct AddressAPI {  
18      func fetchAddresses(userId: String, completion: @escaping ([Address]?, Error?) -> Void) {  
        // Fetch addresses from API  
    }  
19 }  
20
```


#04 - Implementing an API call

```
17 struct AddressAPI {  
18     func fetchAddresses(userId: String, completion: @escaping (Result<[Address], Error>) -> Void)  
    { ... }  
19 }  
20
```

#04 - Implementing an API call

Both suggest using the outdated pattern of a completionHandler.

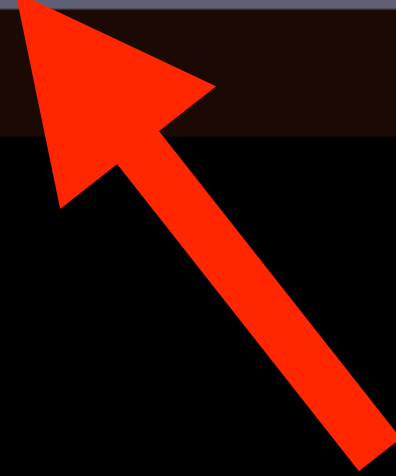
(at least Xcode does suggest using a Result rather than two optionals)

#04 - Implementing an API call

```
17 struct AddressAPI {  
18     func fetchAddresses(userId: String) async -> [Address] {  
19         let url: URL = URL(string: "https://api.example.com/addresses/\(userId)")!  
20  
21         let (data, _) = try! await URLSession.shared.data(from: url)  
22     }  
23 }  
24
```

#04 - Implementing an API call

```
17 struct AddressAPI {  
18     func fetchAddresses(userId: String) async -> [Address] {  
19         let url = URL(string: "https://api.example.com/addresses/\(userId)")!  
20         let data = try await URLSession.shared.data(from: url)  
21     }  
22 }  
23
```



Missing return in instance method

#04 - Implementing an API call

With some hints, both eventually get to the prediction we expect.

Xcode did make a small mistake though.

#05 - Implementing the body of a View

#05 - Implementing the body of a View

```
8  import SwiftUI
9
10 struct AddressView: View {
11
12     @State var address: Address
13
14     var body: some View {
15         VStack {
16             Text(address.street)
17             Text(address.city)
18             Text(address.zipCode)
19         }
20     }
21 }
```

#05 - Implementing the body of a View

```
10 struct Address {  
11     var flskjfsdlf: String  
12     var pbvoivnkljc: String  
13 }
```

#05 - Implementing the body of a View

```
10 struct Address {  
11     var flskjfsdlf: String  
12     var pbvoivnkljc: String  
13 }
```

```
10 struct AddressView: View {  
11  
12     @State var address: Address  
13  
14     var body: some View {  
15         VStack {  
            Text("Address")  
                .font(.title)  
            TextField("Street", text: $address.flskjfsdlf)  
            TextField("City", text: $address.pbvoivnkljc)  
        }  
16     }  
17 }
```

#05 - Implementing the body of a View

```
8  import SwiftUI
9
10 struct AddressView: View {
11
12     @State var address: Address
13
14     var body: some View {
15         VStack {
16             TextField("Address", text: $address.first)
17                 .padding()
18             TextField("City", text: $address.last)
19                 .padding()
20         }
21     }
22 }
```

#05 - Implementing the body of a View

```
10 struct Address {  
11     var sdqzeazqd: String  
12     var jhopjhji: String  
13 }
```

#05 - Implementing the body of a View

```
10 struct Address {  
11     var sdqzeazqd: String  
12     var jhopjhji: String  
13 }
```

```
10 struct AddressView: View {  
11  
12     @State var address: Address  
13  
14     var body: some View {  
15         VStack {  
16             TextField("Address", text: $address.first)  
                .padding()  
                TextField("City", text: $address.last)  
                .padding()  
17         }  
18     }  
19 }  
20
```

#05 - Implementing the body of a View

Xcode's prediction is just plain wrong here...

And this seems to suggest that the prediction doesn't "reason" across files

So what should we conclude?

So what should we conclude?

After using running these tests and having used Predictive Code Completion over the summer, here's my personal take:

- Predictive Code Completion gives equal or worst predictions than Copilot
- Predictive Code Completion takes longer than Copilot to predict
- Predictive Code Completion suffers from UX quirks (no loader)

So what should we conclude?

Now it's not all bad!

- Some use cases, like creating mocks, work relatively well
- Not having to switch to a third-party tool, like VSCode, is really nice
- At times it does manage to make helpful predictions 🙌 🙌 🙌

```
8  import Testing
9
10 @testable import TestingXcodeAI
11
12  struct TestingXcodeAITests {
13
14      @Test func testFilterOutOdds() async throws {
15          let data = [1, 2, 3, 4, 5]
16          #expect(filterOutOdds(data) == [2, 4])
17      }
18
19  }
20
```

```
23 func code() {  
24     let string = "Hellon World!"  
25     string.split(separator: " ")  
26 }  
27
```

M `split(separator:maxSplits:omittingEmptySubsequences:)` >

M `split(maxSplits:omittingEmptySubsequences:whereSeparator:)` >

`split(separator: RegexComponent, maxSplits: Int,
omittingEmptySubsequences: Bool) -> [Substring]`

Returns the longest possible subsequences of the collection, in order, around elements equal to the given separator.

So what should we conclude?

At the end of the day:

- If you're looking for a tool that can do a lot of heavy lifting for you, it doesn't feel that Predictive Code Completion is there yet
- However, it does an honest job at offering suggestions, some of them good, some of them bad, without asking for extra effort from the developer
- If you're on-boarding junior developers, make sure to warn them not to take the predictions at face value
- Hopefully the feature will keep improving over time 🙌

**And if you're really enthusiastic
about AI-assisted coding...**



CURSOR

...you might want to take a
peek at an IDE called Cursor

It offers something quite similar
to what Swift Assist promises

TestingCursorAI

TESTINGCURSORAI

TestingCursorAI

Assets.xcassets

Preview Content

AlertItem.swift

Model.swift

MovieDetailsView.swift

MovieDetailsViewModel.swift

MoviesView.swift

MoviesViewModel.swift

Service.swift

TestingCursorAIApp.swift

TestingCursorAI.xcodeproj

TestingCursorAITests

TestingCursorAITests.swift

MoviesView.swift

TestingCursorAITests.swift

Model.swift

TestingCursorAIApp.swift

MovieDetailsView.swift

Service.swift

TestingCursorAITests > TestingCursorAITests.swift

7

8import XCTest

9

10@testable import TestingCursorAI

11

12| %L to chat, %K to generate

13

OUTLINE

TIMELINE

< 0 0 0

Ln 12, Col 1 Spaces: 4 UTF-8 LF Swift Cursor Tab



That's all Folks!

Thank You! 🤗