



A

Take Flight with AI大模型

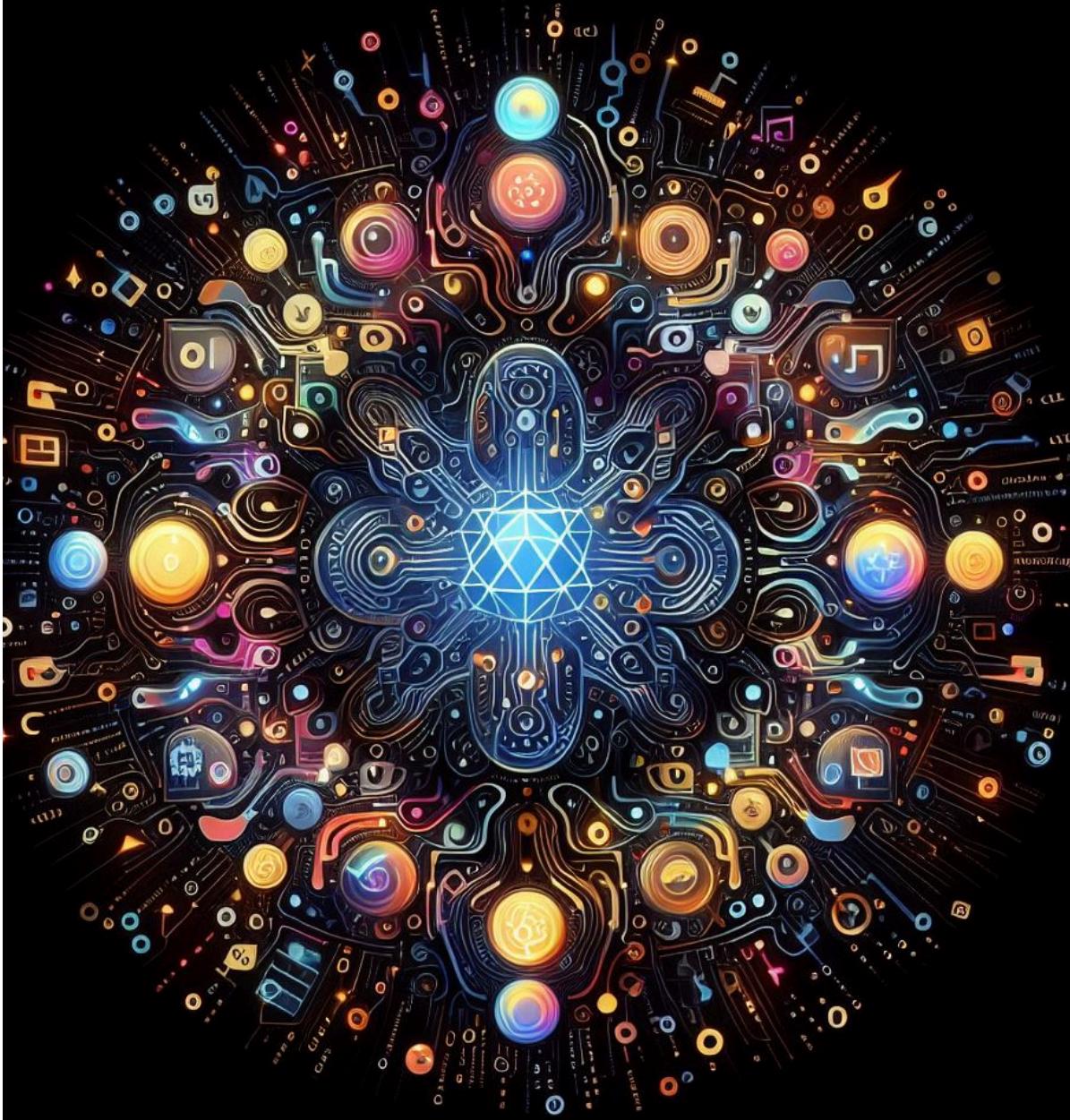
AI大模型辅助编码

Start taking off →

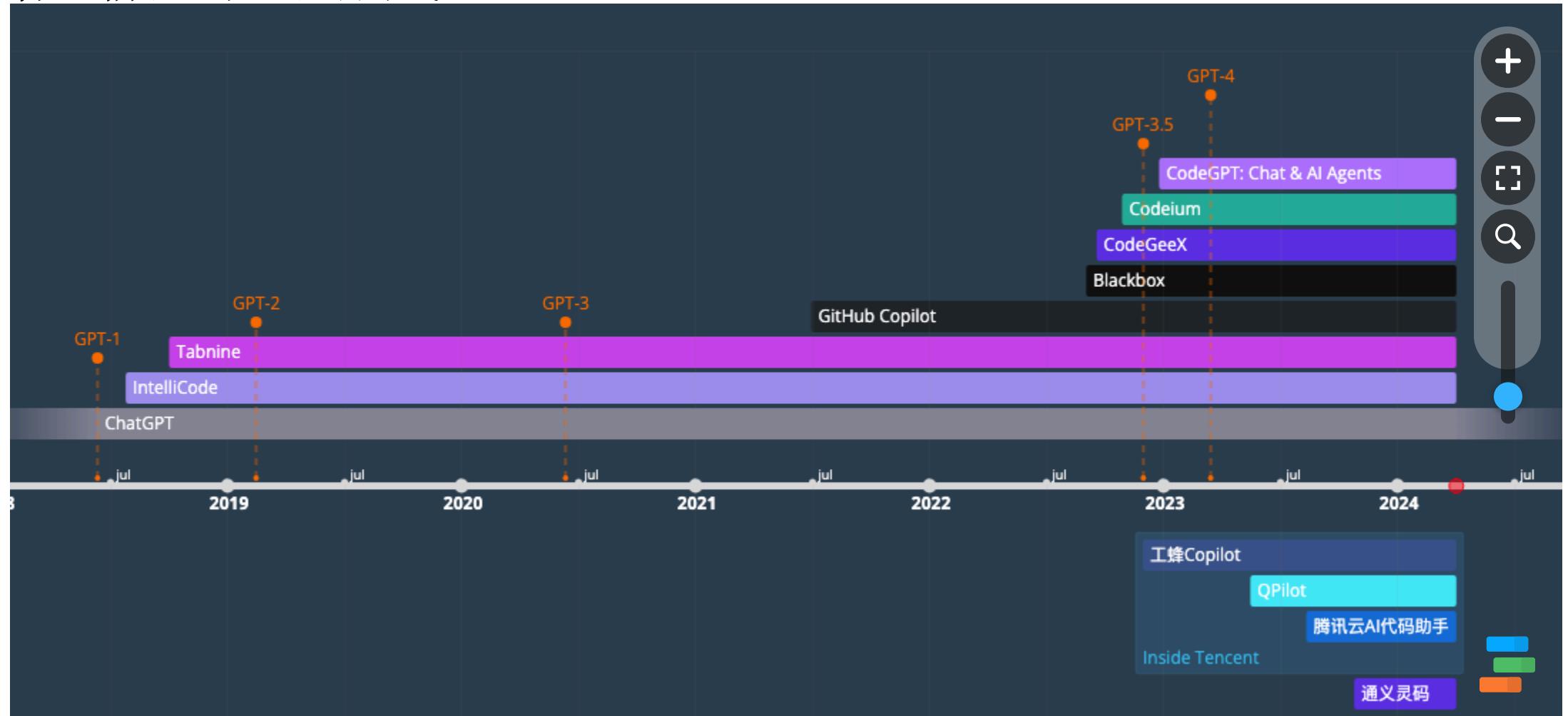
目录

1. take flight with AI大模型

1. 各AI辅助工具的时间线
2. 使用GitHub Copilot的目的&收益
3. AI工具在编码效率上的数据体现
4. 针对AI辅助编码的测评实验
5. AI在编码中是【辅助】还是【干扰】？
6. 优秀的代码补全在编码效率上的实例表现
7. 优秀的代码补全协同技能：注释先行
8. AI助手协同补全的详细拆解
9. 不足分析：补全内容向后关联不足
10. 不足分析：补全结果不精简，相同逻辑块生硬搬迁
11. 不足分析：补全结果不完整的断句
12. 不足分析：重复推荐删除内容 & 跨逻辑块补全
13. 基于AI大模型的CLI应用
14. AI辅助编码的可期待能力



各AI辅助工具的时间线



使用GitHub Copilot的目的&收益

Table 3: Purposes of using GitHub Copilot (results of RQ1.5)

Purpose	Example	Count	%
To help generate code	<i>GitHub CoPilot was able to fill in the code I needed after I wrote the steps in comments (SO #71905508)</i>	26	43.3%
To try out the functionality of Copilot	<i>I use VIM and IntelliJ on a daily basis, and I recently installed VS Code to try out the newest "Copilot Chat" features (SO #76349751)</i>	9	15.0%
To fix bugs	<i>spent a month trying to get it to work on my own and after a week of trying to get chatgpt or copilot to fix the bugs I'm all out of ideas (SO #76266095)</i>	7	11.7%
To improve coding ability	<i>I'm improving my rusty skills lately and saw (in some Copilot suggestions) the question mark operator used as a prefix of variables (SO #74008676)</i>	4	6.7%
To provide ideas for writing code	<i>With the input from jps (AES is actually OK for encrypted tokens, but not signed) and Github Copilot I came up with a working solution using HMAC-SHA256 (SO #72812667)</i>	4	6.7%
For educational purposes	<i>For the past year I have used and taught my students how they could benefit from co-pilot while coding (GitHub #19410)</i>	3	5.0%
To generate code comments	<i>Using ChatGPT and Copilot, I've commented it to understand its functionality. (SO #75624961)</i>	3	5.0%
To check the code	<i>I've checked myself by stepping through, I've checked with GitHub Copilot, I've checked with ChatGPT, and they all say this is correct. (SO #76311798)</i>	2	3.3%
For research purposes	<i>I am working on a scientific study testing how Copilot will effect the academic setting (GitHub #8324)</i>	2	3.3%

最主要的目的：

- 帮助代码生成
- 尝试Copilot的功能
- 修复bug

Table 4: Benefits of using GitHub Copilot (results of RQ2.1)

Benefit	Example	Count	%
Useful code generation	<i>I find myself writing a lot of tests, and Copilot is excellent at helping with writing repetitive tests (GitHub #9282)</i>	28	45.9%
Faster development	<i>I really enjoy using it , it reduce programming time (GitHub #17382)</i>	10	16.4%
Better code quality	<i>it's faster and simpler to your solution (SO #68418725)</i>	7	11.5%
Good adaptation to users' code patterns	<i>Github copilot adapt to your coding practices (SO #69740880)</i>	4	6.6%
Better user experience	<i>Since copilot works totally different compared to all the other products out there, it is a lot more fun to use and does not annoy me like some other AI systems (GitHub #7254)</i>	4	6.6%
Free for students	<i>If you are a student you can sign up for the GitHub Student Pack, which gives a lot of benefits, one being copilot for free (GitHub #31494)</i>	3	4.9%
Powerful code interpretation and conversion functions	<i>Does Copilot have the code explanation feature or something similar? It does! some active members were given beta access. (GitHub #38089)</i>	2	3.3%
Frequent updates to provide more features	<i>Keep in mind that there are updates to the plugin very frequently, so there's still hope (SO #70428218)</i>	1	1.6%
Strong integration capability	<i>Github is supporting more editors (GitHub #6858)</i>	1	1.6%
Ease of study and use	<i>when using this plugin, can study at a relatively low cost (GitHub #8028)</i>	1	1.6%

最主要的收益：

- 有用的代码生成
- 更快的开发
- 更好的代码质量

AI工具在编码效率上的数据体现



在GitHub Copilot官网上，其宣传中的广告文案有提到，Copilot可以帮助用户提升55%的编码效率。



近期有进行过一次评测实验，以上是实验过程中，各个工具在代码采纳率上的表现。

针对AI辅助编码的测评实验

插件	推荐频次 (次/分)	时长 (秒)	推荐次数	次数采纳率 (均值)	次数采纳率 (中值)	采纳字符率 (均值)	采纳字符率 (中值)
工蜂 (gf)	3.42	10944	624	49.85%	50.00%	49.28%	47.24%
腾讯云AI代码助手 (tc)	3.45	8472	487	73.07%	66.35%	67.98%	66.80%
QPilot (qp)	3.40	9896	560	57.50%	59.00%	57.21%	65.00%
Github Copilot (gc)	3.96	8084	534	65.82%	57.87%	75.43%	80.65%
通义灵码 (ty)	4.44	8707	645	59.37%	51.82%	56.69%	51.25%
总计		46103	2850				
12小时48分23秒							

实验前提：

1. 实验人员主动使用相应工具；
2. 实验case侧，同一case由同一实验人员使用不同工具进行评测，保持人员因素的一致性；

AI在编码中是【辅助】还是【干扰】？

AI工具使用过程中，一部分人认为AI工具是【辅助】，另一部分人认为AI工具是【干扰】，那么，AI工具在编码中是【辅助】还是【干扰】？

插件	推荐频次 (次/分)	时长 (秒)	推荐次数	次数采纳率 (均值)	次数采纳率 (中值)	采纳字符率 (均值)	采纳字符率 (中值)
通义灵码 (ty)	4.44	8707	645	59.37%	51.82%	56.69%	51.25%

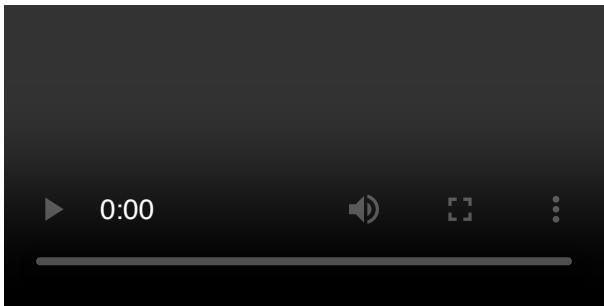
从数据表现上看，通义灵码的表现是：推荐频次最高，次数采纳率和采纳字符率则是倒数第二，如何理解这一数据表现。

其实从实测者角度反馈来说，其表现代表的是，通义灵码有很高的推荐次数，且几乎是一直不停的推荐，但推荐结果对于开发者而言，可采用程度不高，因此整体导致采纳率表现靠后。

这一倒挂的数据表现，其实代表的是，推荐结果不优。

 推荐频次与采纳率之间数据倒挂存在一个临界范围，如果突破这一范围，AI插件的补全行为，不但不会给开发者带来帮助，反而会影响开发者的编码节奏及思路，从而成为干扰。

优秀的代码补全在编码效率上的实例表现

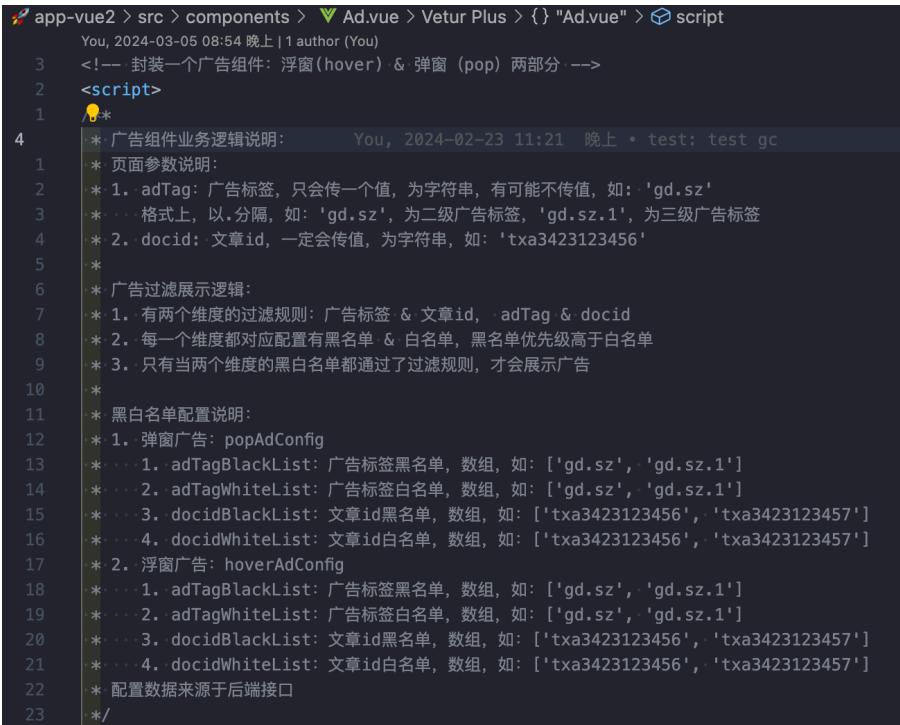


边界良好的补全生成

1. 先生成函数体的注释
2. 再生成函数体

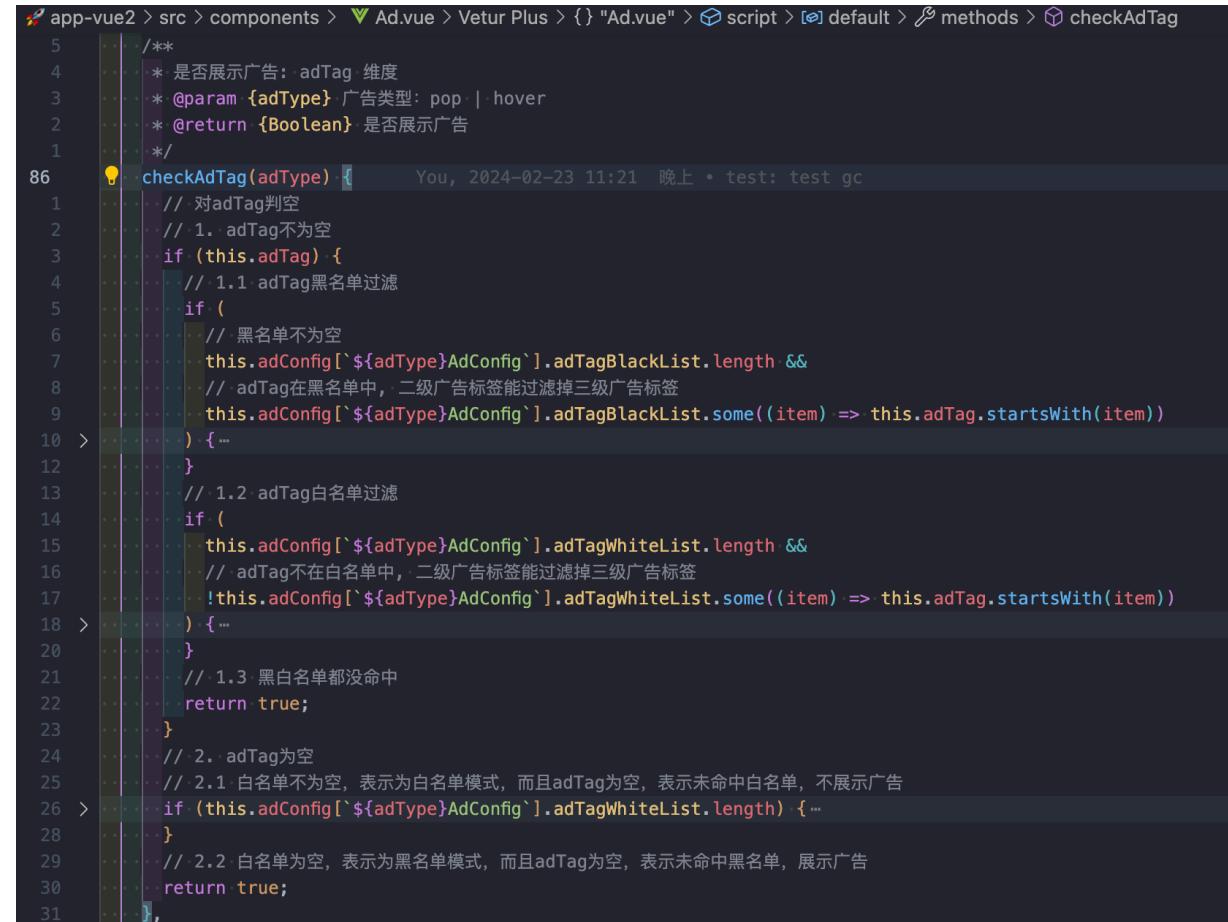
```
/**
 * 是否展示广告: docid 维度
 * @param {adType} 广告类型: pop | hover
 * @return {Boolean} 是否展示广告
 */
checkDocid(adType) {
    // 1. docid黑名单过滤
    if (
        this.adConfig[` ${adType}AdConfig`].docidBlackList.length &&
        this.adConfig[` ${adType}AdConfig`].docidBlackList.includes(this.docid)
    ) {
        return false;
    }
    // 2. docid白名单过滤
    if (
        this.adConfig[` ${adType}AdConfig`].docidWhiteList.length &&
        !this.adConfig[` ${adType}AdConfig`].docidWhiteList.includes(this.docid)
    ) {
        return true;
    }
    // 3. 黑白名单都没命中
    return true;
},
```

优秀的代码补全协同技能：注释先行



```
You, 2024-03-05 08:54 晚上 | 1 author (You)
3 <!-- 封装一个广告组件：浮窗(hover) & 弹窗(pop) 两部分 -->
2 <script>
1 /**
4 * 广告组件业务逻辑说明： You, 2024-02-23 11:21 晚上 . test: test gc
1 * 页面参数说明：
2 * 1. adTag: 广告标签，只会传一个值，为字符串，有可能不传值，如: 'gd.sz'
3 * 格式上，以.分隔，如: 'gd.sz'，为二级广告标签，'gd.sz.1'，为三级广告标签
4 * 2. docid: 文章id，一定会传值，为字符串，如: 'txa3423123456'
5 *
6 * 广告过滤展示逻辑：
7 * 1. 有两个维度的过滤规则：广告标签 & 文章id，adTag & docid
8 * 2. 每一个维度都对应配置有黑名单 & 白名单，黑名单优先级高于白名单
9 * 3. 只有当两个维度的黑白名单都通过了过滤规则，才会展示广告
10 *
11 * 黑白名单配置说明：
12 * 1. 弹窗广告：popAdConfig
13 *   1. adTagBlackList: 广告标签黑名单，数组，如: ['gd.sz', 'gd.sz.1']
14 *   2. adTagWhiteList: 广告标签白名单，数组，如: ['gd.sz', 'gd.sz.1']
15 *   3. docidBlackList: 文章id黑名单，数组，如: ['txa3423123456', 'txa3423123457']
16 *   4. docidWhiteList: 文章id白名单，数组，如: ['txa3423123456', 'txa3423123457']
17 * 2. 浮窗广告：hoverAdConfig
18 *   1. adTagBlackList: 广告标签黑名单，数组，如: ['gd.sz', 'gd.sz.1']
19 *   2. adTagWhiteList: 广告标签白名单，数组，如: ['gd.sz', 'gd.sz.1']
20 *   3. docidBlackList: 文章id黑名单，数组，如: ['txa3423123456', 'txa3423123457']
21 *   4. docidWhiteList: 文章id白名单，数组，如: ['txa3423123456', 'txa3423123457']
22 * 配置数据来源于后端接口
23 */
```

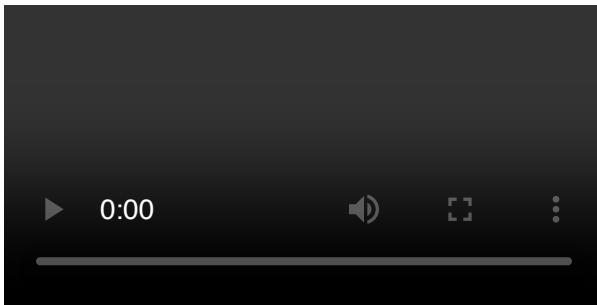
在页面上方对整体文件的逻辑进行注释说明，这将给AI提供一个全局知识背景。



```
5 /**
6 * 是否展示广告：adTag 维度
7 * @param {adType} 广告类型: pop | hover
8 * @return {Boolean} 是否展示广告
9 */
10 checkAdTag(adType) {
11   // 对adTag判断
12   // 1. adTag不为空
13   if (this.adTag) {
14     // 1.1 adTag黑名单过滤
15     if (
16       // 黑名单不为空
17       this.adConfig[$adType].adConfig.adTagBlackList.length &&
18       // adTag在黑名单中，二级广告标签能过滤掉三级广告标签
19       this.adConfig[$adType].adConfig.adTagBlackList.some(item => this.adTag.startsWith(item))
20     ) {
21       // 1.2 adTag白名单过滤
22       if (
23         this.adConfig[$adType].adConfig.adTagWhiteList.length &&
24         // adTag不在白名单中，二级广告标签能过滤掉三级广告标签
25         !this.adConfig[$adType].adConfig.adTagWhiteList.some(item => this.adTag.startsWith(item))
26       ) {
27         // 1.3 黑白名单都没命中
28         return true;
29       }
30     }
31   }
32   // 2. adTag为空
33   // 2.1 白名单不为空，表示为白名单模式，而且adTag为空，表示未命中白名单，不展示广告
34   if (this.adConfig[$adType].adConfig.adTagWhiteList.length) {
35   }
36   // 2.2 白名单为空，表示为黑名单模式，而且adTag为空，表示未命中黑名单，展示广告
37   return true;
38 }
```

实现的函数体内部，会有清晰的注释说明，并分步骤的给出相应段的注释，从注释的结构上，就有一个非常清晰的整理。

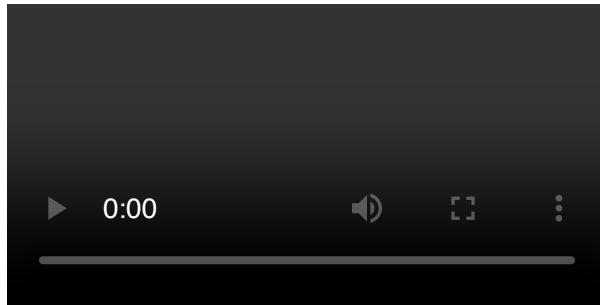
AI助手协同补全的详细拆解



```
*  
* 广告过滤展示逻辑:  
* 1. 有两个维度的过滤规则: 广告标签 & 文章id, adTag & docid  
* 2. 每一个维度都对应配置有黑名单 & 白名单, 黑名单优先级高于白名单  
* 3. 只有当两个维度的黑白名单都通过了过滤规则, 才会展示广告  
*  
* 黑白名单配置说明:  
* 1. 弹窗广告: popAdConfig  
*   1. adTagBlackList: 广告标签黑名单, 数组, 如: ['gd.sz', 'gd.s  
*   2. adTagWhiteList: 广告标签白名单, 数组, 如: ['gd.sz', 'gd.s  
*   3. docidBlackList: 文章id黑名单, 数组, 如: ['txa3423123456'  
*   4. docidWhiteList: 文章id白名单, 数组, 如: ['txa3423123456'  
* 2. 浮窗广告: hoverAdConfig  
*   1. adTagBlackList: 广告标签黑名单, 数组, 如: ['gd.sz', 'gd.s  
*   2. adTagWhiteList: 广告标签白名单, 数组, 如: ['gd.sz', 'gd.s
```

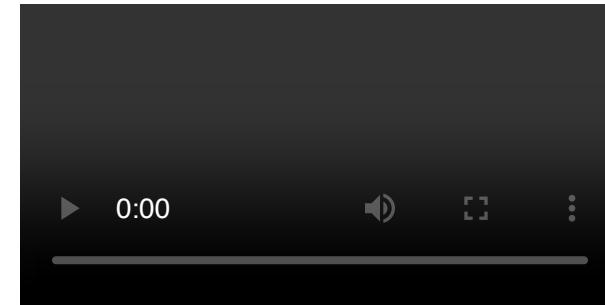
```
/**  
 * 是否展示广告: adTag 维度  
 * @param {adType} 广告类型: pop | hover  
 * @return {Boolean} 是否展示广告  
 */  
checkAdTag(adType) {  
    // 对adTag判空  
    // 1. adTag不为空  
    if (this.adTag) {  
        // 1.1 adTag黑名单过滤  
        if (  
            // 黑名单不为空  
            this.adConfig[` ${adType}AdConfig`].adTagBlackList.length &&  
            // adTag在黑名单中, 二级广告标签能过滤掉三级广告标签  
            this.adConfig[` ${adType}AdConfig`].adTagBlackList.some((item) => this.adTag.startsWith(item))  
        ) {  
            return false;  
        }  
        // 1.2 adTag白名单过滤  
        if (  
            this.adConfig[` ${adType}AdConfig`].adTagWhiteList.length &&  
            // adTag不在白名单中, 二级广告标签能过滤掉三级广告标签  
            !this.adConfig[` ${adType}AdConfig`].adTagWhiteList.some((item) => this.adTag.startsWith(item))  
        ) {  
            return true;  
        }  
        // 1.3 黑白名单都没命中  
        return true;  
    },
```

不足分析：补全内容向后关联不足



其中的返回逻辑段，在后向代码中已实现，补全的结果中不应再次包含。

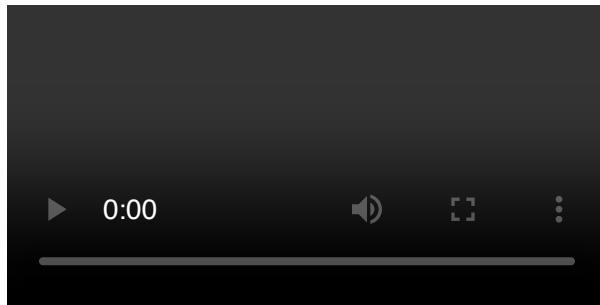
工蜂录屏 (23:00) : <https://drive.weixin.qq.com/s?k=AJEAIQdfAAonyqWesEAL8ACAapAAs>



同样出现重复的返回逻辑段。

腾讯云AI助手录屏 (21:00) : <https://drive.weixin.qq.com/s?k=AJEAIQdfAAo8ppmXQGAL8ACAapAAs>

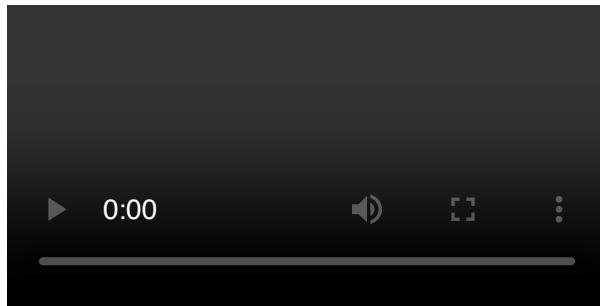
不足分析：补全结果不精简，相同逻辑块生硬搬迁



isShowPopAd逻辑可以通过一行代码直接返回（AI后面在53行是有能力给出正确答案），但实际推出了近10行的代码结果（期间有尝试在多个结果中切换，但未发现最优解），注释再次推荐才给出。（有能力，但未优先）

腾讯云AI助手录屏（27:00）：<https://drive.weixin.qq.com/s?k=AJEAIQdfAAo8ppmXQGAL8ACAapAAs>

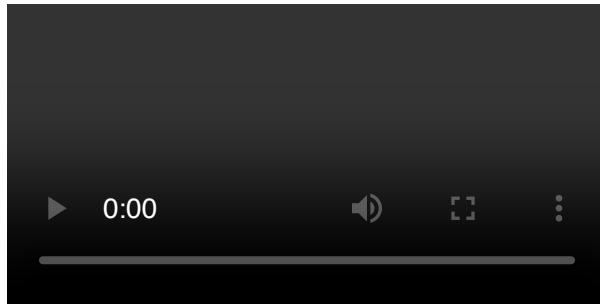
不足分析：补全结果不完整的断句



补全内容的最后一行逻辑不完整，从行中 `this.adConfig[`` 被断开。

QPilot录屏 (15:00) : <https://drive.weixin.qq.com/s?k=AJEAIQdfAAog6kl3tKAL8ACAapAAs>

不足分析：重复推荐删除内容 & 跨逻辑块补全



其中的methods部分，被多次的推出。而这部分的推荐结果曾被多次的删除，即表示开发者的选择，但AI助手仍然会在后续的推荐中，再次推出，这样将给编码工作带来干扰。

另外当前补全的逻辑场景本身就在一个methods的函数中，而其补全的结果中，不但再次出现了methods，重要的是，其已跨出一个大的逻辑块，在上下文的处理过程中，函数这种明确的逻辑边界，最好能分辨并保持在这一逻辑区块下，这样有助于开发者决策判断。

QPilot录屏（29:40）：<https://drive.weixin.qq.com/s?k=AJEAIQdfAAog6kl3tKAL8ACAapAAs>

基于AI大模型的CLI应用

```
# benji @ BENJILLI-MB0 in ~/Documents/tencent/pros/medicalBaike/baike-mono/packages/baike_mini/src/pages/yidian on git:release/2024.03.28-1 o [11:08:09]
C:1
$ gh copilot suggest '在当前目录下查找包含 capsule">的文件，并打印出现的文件名和相应的行号'

Welcome to GitHub Copilot in the CLI!
version 0.5.4-beta (2024-01-04)

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn
and improve. For more information, see https://gh.io/gh-copilot-transparency

? What kind of command can I help you with?
> generic shell command

Suggestion:
grep -rn "capsule\\"" .

? Select an option
> Copy command to clipboard

Command copied to clipboard!

? Select an option
> Exit

# benji @ BENJILLI-MB0 in ~/Documents/tencent/pros/medicalBaike/baike-mono/packages/baike_mini/src/pages/yidian on git:release/2024.03.28-1 o [11:08:25]
$ grep -rn "capsule\\"" .
./baike/drug_detail/components/drugs-float-capsule.vue:2: <div class="drugs-float-capsule">
./node_modules/JSONStream/test/fixtures/all_npm.json:477:[{"id": "capsule", "key": "capsule", "value": {"rev": "19-ad3c9ba0af71a84228e6dd360017f379"}},
./common/components/business/drugs-float-capsule.vue:2: <div class="drugs-float-capsule">
```

AI辅助编码的可期待能力

具有微调能力的AI代码助手产品即将推出。

The image shows a dark-themed user interface for an AI code assistance product. On the left, a purple sidebar contains the text "Coming soon as an add-on" and a large, bold message: "Prefer bespoke? Fine-tune a private, custom model that suggests code based on the best practices and patterns in your repositories." On the right, a white card displays a "Custom models" section with a summary table:

Status	Training data	Training time	Average acceptance rate	Accepted & Retained
Ready	5 repositories	36h 41m	99.4%	99.4%

Below this is a "Model insights" section with a "Languages" chart. The chart shows the distribution of languages included in the trained custom model:

- TypeScript: 90.4%
- Javascript: 9.6%
- Ruby: 9%
- Other: 0.6%

At the bottom of the card, there is a section titled "Preprocessing repositories" with the status "Started 6 minutes ago - Estimated time to finish: 5h 35m" and a log entry: "2023/08/04 08:54:12 Starting preprocessing for 5 repositories...".

感谢观看
Thank you for watching

补充：AI辅助代码解读对比

ifire

解释一下选中的代码

GitHub Copilot

已使用 `this` 引用 >

这段 JavaScript 代码定义了一个名为 `query` 的函数，它用于从 URL 中获取指定的查询参数的值。

下面是这段代码的详细解释：

1. `query = query.replace(/\[\]/, '\[\]').replace(/\[\]/, '\[\]')`: 这行代码将查询参数的名称中的 `[` 和 `]` 字符转义，以便在正则表达式中使用。
2. `var expr = '[(?&)]+' + query + '=([^#]*)'`: 这行代码创建了一个正则表达式的字符串，用于匹配 URL 中的查询参数。这个正则表达式会匹配以 `?=` 或 `&` 开始，然后是查询参数的名称，然后是 `=`，然后是查询参数的值，直到遇到 `&` 或 `#` 为止。
3. `var regex = new RegExp(expr)`: 这行代码创建了一个新的正则表达式对象。
4. `url = (url || window.location.href)`: 这行代码获取要搜索的 URL，如果没有提供 URL 参数，那么就使用当前窗口的 URL。
5. `var results = regex.exec(url)`: 这行代码使用正则表达式在 URL 中搜索查询参数。
6. `if (results !== null) { return decodeURIComponent(results[1].replace(/\+/g, ' ')) }`: 这行代码检查搜索结果。如果找到了查询参数，那么就返回查询参数的值。查询参数的值是 URL 解码的，并且所有的 `+` 字符都被替换为空格。如果没有找到查询参数，那么就返回空字符串。
7. `else { return '' }`: 这行代码如果未找到查询参数，则返回空字符串。
8. `this.loginWithWechat()`: 登录方法调用。
9. `//登录`: 备注。
10. `function loginWithWechat () {`: 定义一个名为 `loginWithWechat` 的方法。
11. `var url = window.location.href`: 赋值给 `url` 变量。
12. `var code = this.query("code", url)`: 调用 `this.query` 方法获取 `code` 参数。
13. `var state = this.query("state", url)`: 调用 `this.query` 方法获取 `state` 参数。
14. `if (!code && !state) {`: 检查 `code` 和 `state` 是否都不存在。
15. `var redirect = "https://open.weixin.qq.com/connect/oauth2/authorize?appid=" + appid + "&redirect_uri=" + encodeURIComponent(url) + "&response_type=code&scope=snsapi_base#wechat_redirect"`: 构建重定向 URL。
16. `location.replace(redirect)`: 使用 `location.replace` 方法跳转到重定向 URL。
17. `}`: 方法结束。
18. `else {`: 如果 `code` 存在。
19. `if (code) {`: 检查 `code` 是否存在。
20. `ajax.post('LoginWithWeixin', { code: code, appid: appid }, function (err) {`: 使用 `ajax.post` 方法发送 POST 请求。
21. `if (err) {`: 检查请求是否失败。
22. `if (err) {`: 如果失败，返回错误信息。
23. `console.log(res)`: 打印响应结果。
24. `this.token = res.token`: 将响应中的 `token` 存储在 `this.token` 中。

GitHub Copilot 在代码解读上，对于代码的理解能力，当前实验的 case 表现上，还是有更好一些的表现，交互上也更为合理。

Benjili

解释一下这段代码

工蜂Copilot

这段代码是一个名为 `query` 的函数，它接受两个参数：`query` 和 `url`。函数的主要目的是从给定的 `url` 中提取特定的查询参数 (`query`) 对应的值。

代码的主要步骤是：

1. 首先，将 `query` 参数中的字符串中的方括号 (`[]`) 替换为其实现版本的 `'['` 和 `']'`，以避免与正则表达式中的字符集合符号 (`[]`) 发生冲突。
2. 定义一个名为 `expr` 的字符串，用于存储用于提取查询参数值的正则表达式。该正则表达式包含查询参数 (`query`)，用于匹配 URL 中的查询参数值。
3. 使用 `expr` 字符串创建一个名为 `regex` 的正则表达式对象。
4. 如果没有提供 `url` 参数，则使用浏览器当前的 URL。
5. 使用 `regex.exec(url)` 提取 URL 中查询参数值的结果。
6. 如果结果不为空 (`results !== null`)，则解码并返回查询参数值。
7. 如果结果为空 (`results === null`)，则返回空字符串。

总之，这个函数可以从给定的 URL 中提取特定查询参数对应的值。

评分：★★★★★

重新生成

```
function query (query, url) {    igeraldchen, 2020-07-15 04:57
  query = query.replace(/\[\]/, '\[\]').replace(/\[\]/, '\[\]')
  var expr = '[(?&)]+' + query + '=([^#]*)'
  var regex = new RegExp(expr)
  url = (url || window.location.href)
  var results = regex.exec(url)
  if (results !== null) {
    return decodeURIComponent(results[1].replace(/\+/g, ' '))
  } else {
    return ''
  }
}

this.loginWithWechat()

//登录
function loginWithWechat () {
  var url = window.location.href
  var code = this.query("code", url)
  var state = this.query("state", url)
  if (!code && !state) {
    var redirect = "https://open.weixin.qq.com/connect/oauth2/authorize?appid=" + appid + "&redirect_uri=" + encodeURIComponent(url) + "&response_type=code&scope=snsapi_base#wechat_redirect"
    location.replace(redirect)
  } else {
    if (code) {
      ajax.post('LoginWithWeixin', { code: code, appid: appid }, function (err) {
        if (err) {
          console.log(res)
          this.token = res.token
        }
      })
    }
  }
}
```

工蜂的代码解读功能，从当前的测试 case 来看，其表现上，还有一定的提升空间，交互上也可以向多轮对话，以及与编辑器中代码交互的方向靠拢。

补充：AI提供代码review的能力

- 能力要求：
 - 充足的代码规范、漏洞、安全、编程最佳实践等知识，最好能具备一定的业务知识（独立微调，或是外挂）
- 切入角度：
 - 集成到相关的review流水线，实现自动化
- 最佳工具：
 - 目前还没有发现非常合意的工具，原因：review带有很强的业务特色，需要嵌入到业务内部来review，这样才有深度

review会因各自工程的规范，而产生差异，无法全面的能用 使用过的是工蜂的AI评审，eg.: 示例MR

The screenshot shows a code review interface for a file named 'Copilot'. The code is written in JavaScript and handles image saving. The interface includes a toolbar with icons for like, dislike, and other actions, and a preview area showing the review range from line 68 to 105. The code editor shows the following snippet:

```
68 + saveImg(url) {  
69 +     wx.getImageInfo({  
70 +         src: url,  
71 +         success: (res) => {  
72 +             const { path } = res;  
73 +             wx.saveImageToPhotosAlbum({  
74 +                 filePath: path,  
75 +                 success: () => {  
76 +                     showToast({  
77 +                         title: '已成功保存到相册，快去分享吧！',  
78 +                         icon: 'none',  
79 +                         duration: 2000,  
80 +                     });  
81 +                 },  
82 +                 fail: (e) => {  
83 +                     showToast({  
84 +                         title: '图片保存失败',  
85 +                         icon: 'none',  
86 +                         duration: 2000,  
87 +                     });  
88 +                     // 非用户取消  
89 +                     if (e && eerrMsg !== 'saveImageToPhotosAlbum:fail cancel') {  
90 +                         reportBusinessMonitor('saveImageToPhotosAlbum.error', e.message || {});  
91 +                     }  
92 +                 },  
93 +             });  
94 +         },  
95 +         fail: (e) => {  
96 +             console.log(e);  
97 +             showToast({  
98 +                 title: '图片保存失败',  
99 +                 icon: 'none',  
100 +                 duration: 2000,  
101 +             });  
102 +             reportBusinessMonitor('getImageInfo.error', (e && e.message) || {});  
103 +         },  
104 +     });  
105 + }
```

Annotations above the code highlight two issues:

- 1. 在`wx.getImageInfo()`、`wx.saveImageToPhotosAlbum()`和`console.log(e)`等异步函数中直接使用了箭头函数，这可能会导致this上下文丢失，从而影响到函数内部的操作。建议在这些地方使用普通函数表达式或者bind方法来保证正确的上下文。
- 2. 当调用`reportBusinessMonitor()`函数时，没有传入错误对象的详细信息，这可能会使得调试和问题定位变得困难。建议在这里添加更多的信息，比如错误发生的代码行号、错误类型等。

