

# 基于 ViperGPT 框架的功能改进

## 摘要

本次实验为基于 ViperGPT 框架的功能改进。ViperGPT 简单来说就是让代码生成模型根据用户输入的问题以及预先设计好的 API 接口，生成可以解决用户问题的代码，然后再执行代码，得到答案。这种简单的方法不需要进一步的训练，并在多种复杂的视觉任务上取得了优异的结果。然而目前 ViperGPT 的性能还有很大的提升空间，并且其功能还是有所欠缺，许多人类生活中经常碰到的视觉任务，如图片文字提取、图片事物识别等，它还不能解决。因此，在 ViperGPT 的源代码上进行改进，替换掉性能较差的模型，设计新的 API 接口，添加对应的模型，修改 prompt，让 ViperGPT 能够更好地适应新的挑战。

**关键词：**ViperGPT；代码生成；图片文字提取；图片事物识别；

## 1 引言

回答视觉查询是一种计算机视觉领域的任务，其目标是使计算机系统能够理解和回应对图像或视频提出的自然语言问题。这一任务涵盖了多个方面，包括图像理解、自然语言处理和推理等领域的交叉。目前的主要方法为端到端训练的模型，这种模型的做法是将视觉处理和推理两个步骤都让模型一步解决，限制了可解释性和泛化能力。学习模块化程序是一种很有前途的替代方案，但由于难以同时学习程序和模块，已被证明具有挑战性。

本次工作使用的 ViperGPT 框架，它利用代码生成模型将视觉和语言模型组合成子程序，以生成任何查询的结果。ViperGPT 利用提供的 API 访问可用的模块，并通过生成稍后执行的 Python 代码来组合它们。这种方法可解性高，泛化能力强，性能优异。

然而，目前的 ViperGPT 使用的视觉处理性能并不是最好的，限制了它的能力，因此可以将其性能较差模型更换为更好的模型。此外，由于已经设计好的接口并不能处理所有的视觉任务，比如图片文字提取、图片事物识别等，设计新的 API 接口，添加对应的模型修改 prompt，让 ViperGPT 能够适应新的挑战。

## 2 相关工作

### 2.1 模块化视觉

模块化视觉。ViperGPT 的灵感来源于神经模块网络 [2,9]，他们认为复杂的视觉任务基本上是由组合部分构成的，并建议将它们分解为原子感知单元。这种视觉推理过程已经被各种研究所探讨 [10,19]。后续的工作集中在通过将推理与感知分离来明确地思考组合，涉及神

经符号方法 [7,9]。这些方法在想法上与 ViperGPT 相似，但需要以程序的形式提供监督，并通过端到端训练感知模块，这使它们不能泛化到不同的领域。

由于使用上述方法的实际困难，该领域主要转向端到端的一体化模型 [1,8]，这类模型目前取得了最优异的结果。最近的其他研究表明 [18,20]，大型预训练模型可以一起取得很好的效果，但是需要手动指定模型组合的特定方式。目前，该领域的研究兴趣激增，导致了一系列与之相关的手稿出现在 arXiv 上，这些手稿使用大型语言模型 (LLMs) 进行模块集成。在自然语言处理领域，它们旨在使用外部工具 [14,16]，或者使用 Codex 进行结构化推理 [13,17]。与这些方法不同的是，ViperGPT 直接生成不受限制的 Python 代码，这更加灵活，使其能够展示更高级的新兴能力，如控制流和数学运算。此外，使用 Python 允许 ViperGPT 利用 Codex 通过在互联网上进行大规模训练而学到的强大先验知识。

## 2.2 预训练模型

ViperGPT 使用的感知和外部知识模块包括：GLIP [12] 用于目标检测，X-VLM [21] 用于文本-图像相似性，MiDaS [15] 用于深度估计，GPT-3 [4] 用于外部知识，以及 BLIP-2 [11] 用于简单的视觉查询。

## 3 本文方法

### 3.1 本文方法概述

ViperGPT 使用了符号表示法，即给定一个视觉输入  $x$  和一个关于其内容的文本查询  $q$ ，首先通过给定查询生成器 合成一个程序  $z = (q)$ 。然后将执行引擎  $r = (x, z)$  应用于在输入  $x$  上执行程序  $z$  并生成结果  $r$ 。该框架具有灵活性，支持图像或视频作为输入  $x$ ，问题或描述作为查询  $q$ ，以及任何类型的输出  $r$ （例如文本或图像裁剪）。虽然先前的工作将程序表示为图，如语法树或依赖图，但 ViperGPT 直接通过 Python 代码表示程序  $z$  的类别，使程序能够充分利用现代编程语言所提供的表达能力和功能。ViperGPT 的框架如图 1 所示：

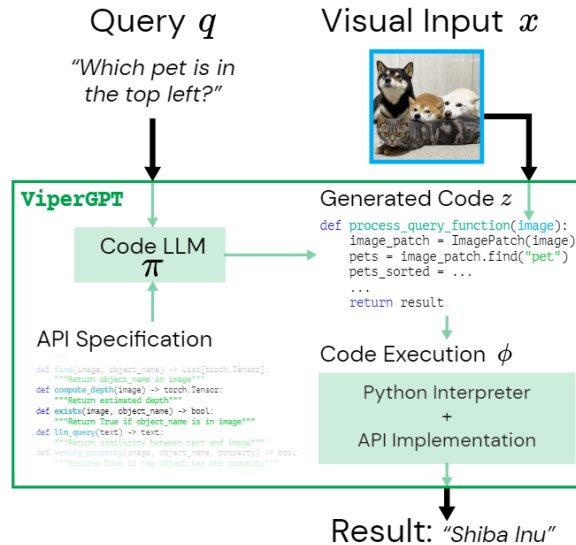


图 1. ViperGPT 框架图

### 3.2 程序生成

ViperGPT 利用语言模型 (LLMs) 进行代码生成，以实例化程序生成器，将视觉和语言模块结合起来。LLMs 将标记化的代码序列 (“提示”) 作为输入，并自回归地预测后续标记。使用的具体模型为 Codex [5]，该模型在代码生成任务上表现出色。由于 ViperGPT 用 LLM 替代了 的优化过程，消除了为程序生成需要特定任务的训练的需求。使用 Codex 作为程序生成器，并直接在 Python 中生成代码，使其能够充分利用在互联网上进行的规模化训练，其 Python 代码丰富多样。为了以这种方式利用 LLMs，我们需要定义一个提示，该提示将对组成和调用这些模块所需的程序  $z$  进行抽样。提示包括一个应用程序编程接口 (API)，将其作为 LLM 输入上下文的一部分提供。LLM 的最终输入是一个代码文本序列，包括 API 规范，以及相应的问题查询。预期的输出是一个 Python 函数定义作为字符串，然后将其编译和执行。

### 3.3 模块及其 API

ViperGPT 提供的提示包含了不同感知和知识模块的 API，例如目标检测、深度估计或语言模型查询。通过这个提示，我们发现 LLMs 能够从查询  $q$  中归纳出正确的程序  $z$ 。提供的 API 定义了两个全局类 ImagePatch 和 VideoSegment，分别表示图像块和视频段。每个模块都被实现为一个类方法，该方法内部调用一个预训练模型来计算结果。例如，ImagePatch 的 computedepth 方法返回图像块中像素的中位 (相对) 深度的估计；使用 MiDaS [15] 等最先进的大规模模型来实现这一功能。API 为其定义的每个方法规定了输入和输出类型，以及用自然语言解释这些函数目的的文档字符串。与大多数 API 类似，它还提供了示例，展示了如何使用这些类及其函数，类似于上下文学习中的查询代码对。Codex 的输入不包含 API 的完整实现。相反，它给出了 API 的规范，包括函数签名和文档字符串。去除实现细节有两方面好处。首先，LLM 上下文窗口的大小有限 [6]，无法包含整个实现，此外，这种抽象使得代码生成与对模块实现所做的更改无关。端到端感知模块在正确的地方使用时非常出色，而 ViperGPT 强烈依赖于它们。类似于认知科学中的双系统模型，我们认为生成的程序 (System 2 - 分析型) 应该被用于将需要多步推理的任务分解为更简单的组件，其中端到端感知模块 (System 1 - 模式识别) 是最有效的方法。通过将端到端模块组合成程序，ViperGPT 为深度学习带来了串行处理的 System 2 能力 [3]。

### 3.4 程序执行

ViperGPT 通过使用 Python 解释器与大型预训练模型实现的模块结合，提供了一个简单而高效的替代方案。Python 解释器实现了逻辑操作，而预训练模型实现了感知操作。程序在 Python 解释器中运行，因此其执行是一个简单的 Python 调用，这意味着它可以利用所有内置的 Python 函数，比如 sort；控制流工具，比如 for 或 if/else；以及模块，比如 datetime 或 math。值得注意的是，它不需要定制解释器。完全 Python 化实现的另一个优势是与许多现有工具的兼容性，如 PyTorch。

## 4 复现细节

### 4.1 与已有开源代码对比

经过研究，发现 ViperGPT 有以下三点不足，一是使用的感知模型并不都是效果最好的模型，如 Glip，而 ViperGPT 的性能又十分依赖于感知模块的效果，所以第一个创新点为替换性能更好的模型，在此次实验中将原来的 Glip 模型替换为 GroungingDINO 模型；二是设计的 API 并不能解决所有的问题查询，许多人类生活中经常碰到的视觉任务，如图片文字提取、图片事物识别等，它还不能解决，所以第二个创新点是为其设计新的 API，使其具有图片文字提取、图片事物识别功能，同时配置所需模型，包括 EasyOCR 和 RAM [22]，修改 prompt，让 ViperGPT 能够解决新的任务。三是受限于代码生成模型的性能，其不能处理较为复杂的问题，如“找出骑摩托车没有戴头盔的人”这个问题，生成的代码往往只能找出“骑模态车的人”或“戴头盔的人”，而考虑到这种多属性检测在实际生活中有比较多的应用，因此基于生成的简单代码进行改进，使其能够实现多属性检测功能，同时为该功能设计了交互界面。

为了评估模型的性能，从 MME [6] 数据集里面抽取了 100 个问题和对应的图片，涵盖颜色、位置、存在、技术和 OCR 等多个类别。MME 专为评估多模态大语言模型性能而设计，包含 1000 多张图像以及相应的问题。

### 4.2 实验环境搭建

本次实验所用软硬件环境为：python==3.10，pytorch==1.13.0，CUDA==11.6，GPU：A100。由于 ViperGPT 已经开源，按照其开源仓库配置环境。首先通过指令克隆仓库，配置环境，如图 2 所示。因为 ViperGPT 需要调用感知模型，我们需要预先下载所有可能调用到的模型，包括了 Glip，GroungDINO，MiDaS，BLIP-2，X-VLM 等，此步骤需要占用较长的时间。

```
git clone --recurse-submodules https://github.com/cvlab-columbia/viper.git
cd viper
export PATH=/usr/local/cuda/bin:$PATH
bash setup.sh # This may take a while. Make sure the vipergpt environment is active
cd GLIP
python setup.py clean --all build develop --user
cd ..
echo YOUR_OPENAI_API_KEY_HERE > api.key
```

图 2. 配置环境指令

### 4.3 图片文字提取功能使用说明

新的 API 包括图片文字提取、图片事物识别功能，以图片文字提取功能为例，首先准备过程，包括导入相关包和加载所需模型，指令为如图 3 所示。此过程需要较长时间加载模型。

```
from main_simple_lib import *
```

图 3. 准备过程

代码生成过程如图 4 所示，其中包含了图片地址 path，加载图片 load\_image，问题查询 query，以及代码生成 get\_code。在此过程，问题查询与预先设计的提示一起输入到代码生成模型，生成相应的可执行代码。

```
[2]: path = "/public26_data/lsx/test/IWE_5/ocr/0001.jpg"
     im = load_image(path)
     query = 'Is the word Yuyuan in the picture?'
     show_single_image(im)
     code = get_code(query)
```



```
def execute_command(image) -> str:
1  image_patch = ImagePatch(image)
2  word = image_patch.ocr()
3  target = 'Yuyuan'
4  joined_string = ''.join(word)
5  if target in joined_string:
6      return 'yes'
7  else:
8      return 'no'
```

图 4. 代码生成


代码执行过程如图 5 所示，只需运行 execute\_code 代码，生成的代码将会自动执行。每一步的执行结果都会显示出来，以便检查是否有问题，更容易定位和解决潜在的问题。

```
execute_code(code, im, show_intermediate_steps=True)
```

```
def execute_command(image) -> str:
1  image_patch = ImagePatch(image)
2  word = image_patch.ocr()
3  target = 'Yuyuan'
4  joined_string = ''.join(word)
5  if target in joined_string:
6      return 'yes'
7  else:
8      return 'no'
```

---

```
image_patch =
```



---

```
word[0] = 西
word[1] = 愚园路
word[3] = 315
word[4] = 309
word[6] = Yuyuan Rd.
word[7] = E
```

---

```
joined_string = 西愚园路东315309Yuyuan Rd. E
```

---

```
target in joined_string = True
```

---

```
Final Result
```

```
Result = yes
'yes'
```

图 5. 代码执行



4.4 多属性检测功能使用说明

多属性检测如图 6所示，其中检测主体为必须填入部分，如“person”，相关属性和不具备属性可填可不填，展示的实例为检测“骑摩托车未戴头盔的人”这个目标。

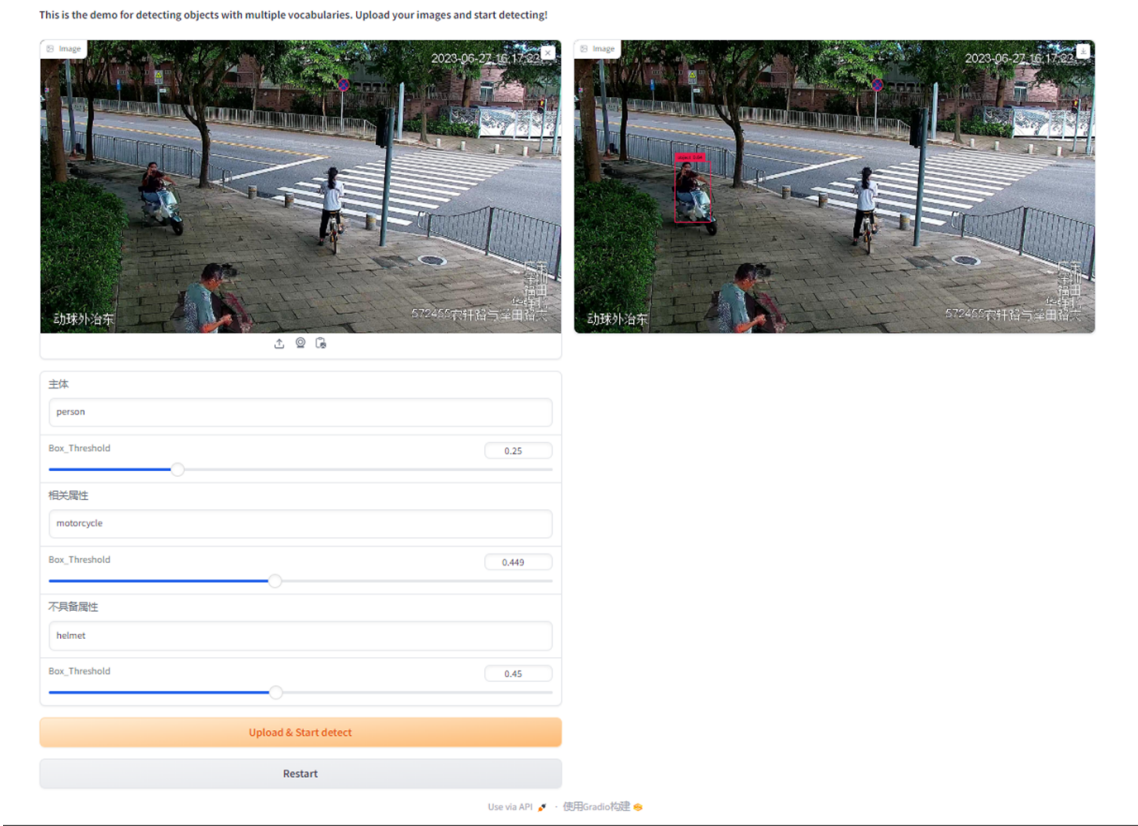


图 6. 多属性检测

5 实验结果分析

实验结果如表 1所示，在将检测模型更换为 GroundingDino 后，对于物体存在性和位置的判断表现略有提升。然而，在计数性能方面出现了显著下降，可能是 GroundingDino 多次检测同一物体导致的。在 OCR 方面，原始的 ViperGPR 不具备 OCR 功能，其正确率为 10%，而在加入 OCR 模块后，正确率提高至 50%。总体而言，改进后的 Viper 相较于原版性能提高了 8%。

表 1. 评估结果

问题类别	ViperGPT(原)	ViperGPT(GD+EasyOCR)
Color	0.80	0.80
Position	0.70	0.75
Existence	0.85	0.95
Count	0.95	0.8
Ocr	0.1	0.5
All	0.68	0.76

## 6 总结与展望

本次实验基于 ViperGPT 框架进行功能改进，以提升在多种复杂的视觉任务中的性能。ViperGPT 这种简单而高效的方法在多种复杂的视觉任务中表现出色，无需进一步的训练即可取得良好结果。然而，目前 ViperGPT 在性能和功能上仍存在提升的空间，尤其在一些常见的人类生活中经常遇到的视觉任务上，例如图片文字提取和图片事物识别等方面，其表现尚不尽如人意。

通过设计创新，包括替换性能较差的模型，设计新的 API 接口，添加相应的模型，以及修改 prompt，使 ViperGPT 更好地适应新的挑战。具体而言，替换了检测模型为 GroundingDino，通过评估结果发现在物体存在性和位置的判断上略有提升，但在计数性能方面存在显著下降。另外，在 OCR 方面，我们增加了 OCR 模块，将原始的 ViperGPT 的 OCR 正确率从 10% 提升至 50%。总体而言，改进后的 Viper 相较于原版性能提高了 8%。

未来，将继续努力改进 ViperGPT 的性能和功能，以使其更好地适应各种视觉任务。具体而言，计划在以下几个方面进行改进：

性能优化：进一步优化感知模型，用性能更好的模型替换初始感知模型，确保在各类任务中都能取得更好的表现。

功能扩展：添加更多针对人类生活中常见视觉任务的功能，例如增强图片文字提取、图片事物识别等模块，使 ViperGPT 在更广泛的应用场景中得到应用。

用户体验改善：提高用户与 ViperGPT 的交互体验，包括更友好的提示信息、更直观的输入方式等，使用户更容易理解和操作。

通过这些改进，我们期望 ViperGPT 能够成为一个更加全面、灵活且性能卓越的视觉任务解决方案，为用户提供更好的服务和体验。

## 参考文献

- [1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.
- [2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.
- [3] Yoshua Bengio. The consciousness prior. *Cornell University - arXiv, Cornell University - arXiv*, Sep 2017.
- [4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

- [5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [6] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. Mme: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394*, 2023.
- [7] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [8] Ziniu Hu, Ahmet Iscen, Chen Sun, Zirui Wang, Kai-Wei Chang, Yizhou Sun, Cordelia Schmid, David A. Ross, and Alireza Fathi. Reveal: Retrieval-augmented visual-language pre-training with multi-source multimodal knowledge memory. Dec 2022.
- [9] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [10] SeungWook Kim, Makarand Tapaswi, and Sanja Fidler. Visual reasoning by progressive module networks. *arXiv: Computer Vision and Pattern Recognition, arXiv: Computer Vision and Pattern Recognition*, Jun 2018.
- [11] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models.
- [12] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. Grounded language-image pre-training. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2022.
- [13] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. Oct 2022.
- [14] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. May 2022.
- [15] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1623–1637, 2020.
- [16] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. Feb 2023.



- [17] Xingyao Wang, Sha Li, and Heng Ji. Code4struct: Code generation for few-shot structured prediction from natural language. Oct 2022.
- [18] Zhenhailong Wang, Manling Li, Ruochen Xu, Luowei Zhou, Jie Lei, Xudong Lin, Shuohang Wang, Ziyi Yang, Chenguang Zhu, Derek Hoiem, Shih-Fu Chang, Mohit Bansal, and Heng Ji. Language models with image descriptors are strong few-shot video-language learners.
- [19] Spencer Whitehead, Hui Wu, Heng Ji, Rogerio Feris, and Kate Saenko. Separating skills and concepts for novel visual question answering. *arXiv: Computer Vision and Pattern Recognition, arXiv: Computer Vision and Pattern Recognition*, Jul 2021.
- [20] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language.
- [21] Yan Zeng, Xinsong Zhang, and Hang Li. Multi-grained vision language pre-training: Aligning texts with visual concepts.
- [22] Youcai Zhang, Xinyu Huang, Jinyu Ma, Zhaoyang Li, Zhaochuan Luo, Yanchun Xie, Yuzhuo Qin, Tong Luo, Yaqian Li, Shilong Liu, et al. Recognize anything: A strong image tagging model. *arXiv preprint arXiv:2306.03514*, 2023.