GPTCelltype: Reference-free and cost-effective automated cell type annotation with GPT-4 in single-cell RNA-seq analysis

Wenpin Hou[1,*], Zhicheng Ji[2,*]
[1]Department of Biostatistics, Columbia University Mailman School of Public Health Health
[2]Department of Biostatistics and Bioinformatics, Duke University School of Medicine
* corresponding authors

Introductions

Cell type annotation is an essential step in single-cell RNA-seq analysis. However, it is a time-consuming process that often requires expertise in collecting canonical marker genes and manually annotating cell types. Automated cell type annotation methods typically require the acquisition of high-quality reference datasets and the development of additional pipelines. We demonstrated that GPT-4, a highly potent large language model, can automatically and accurately annotate cell types by utilizing marker gene information generated from standard single-cell RNA-seq analysis pipelines in this manuscript. We developed this software, **GPTCelltype**, to provide an automated cell type annotation approach using GPT-4 for single-cell RNA-seq analysis.

Installation

GPTCelltype can be installed by following this instruction on Github.

```
remotes::install_github("Winnie09/GPTCelltype")
```

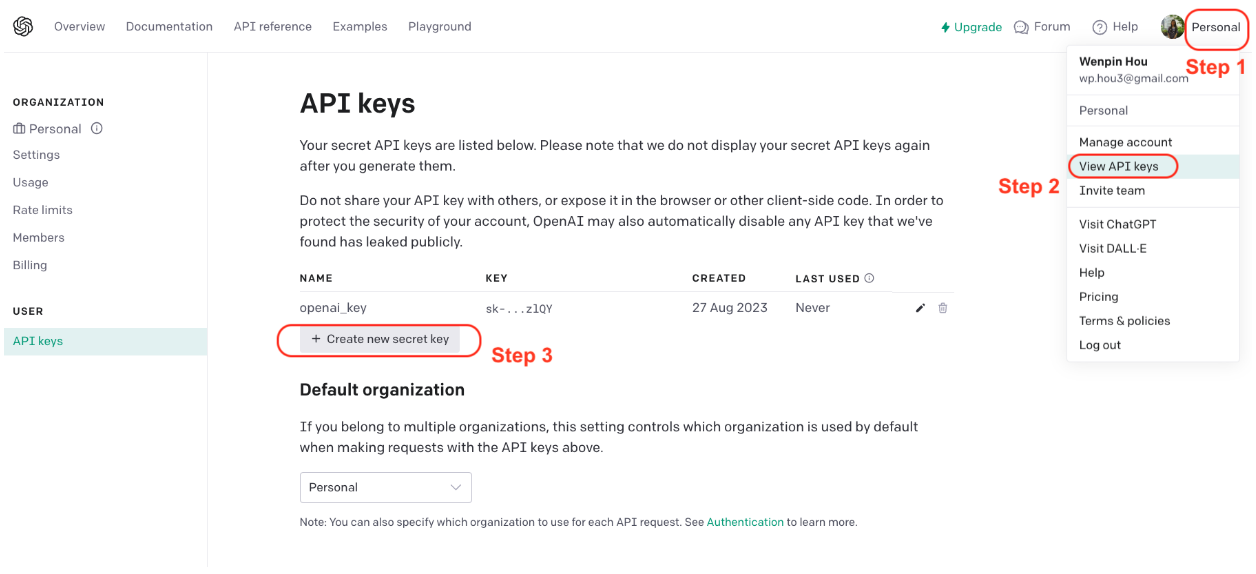
GPTCelltype depends on the R package openai. Please make sure it is installed as well.

```
install.packages("openai")
```

Set up OpenAI API key as an environment variable

GPTCelltype integrates the OpenAI API into the software. To connect to OpenAI API, a secret API key is required. To avoid the risk of exposing the API key or committing the key to browsers, users need to set up the API key as a system environment variable before running GPTCelltype. If the API key is provided, cell type annotations are returned. Otherwise, if the API key is not provided, the output from GPTCelltype is the prompt itself which users can further used to communicate with the GPT chatbot.

You can generate your API key in your OpenAI account webpage: log in to OpenAI, click on “Personal” on the upper right corner, click on “View API keys” in the break-down list, and then click on “Create new secret key” which directs you to the API key page. Copy the key and paste it on a note for further use. Avoid sharing your API key with others or uploading it to public spaces. Make sure it’s not visible in browsers or any client-side scripts.



API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically disable any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED
openai_key	sk-...zlQY	27 Aug 2023	Never

Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

Personal

Note: You can also specify which organization to use for each API request. See Authentication to learn more.

Set up the API key as a system environment variable before running GPTCelltype.

```r
Sys.setenv(OPENAI_API_KEY = 'your_openai_API_key')
```

Run GPTCelltype

First of all, please load the packages.

```r
library(GPTCelltype)
library(openai)
```

We demonstrate how to run GPTCelltype as follows. The main function is `gptcelltype()`. It can annotate cell types by OpenAI GPT models in a Seurat pipeline or with a custom gene list. If `gptcelltype()` is used in a Seurat pipeline, Seurat `FindAllMarkers()` function needs to be run first and the differential gene table generated by Seurat will serve as the input. If the input is a custom list of genes, one cell type is identified for each element in the list.

Among the input arguments, `input` can either be the differential gene table returned by Seurat `FindAllMarkers()` function, or a list of genes. `tissuename` (optional) is a tissue name. `model` is a valid GPT-4 or GPT-3.5 model name listed on Models page. Default is 'gpt-4'. `topgenenumber` is the number of top differential genes to be used when the input is Seurat differential genes. The output is a vector of cell types.

Example 1: Seurat object as input GPTCelltype integrates seamlessly with the Seurat pipeline. It can take an Seurat object as input, if the Seurat object has marker genes information. Specifially, this can be achieved after running the Seurat function `FindAllMarkers()`. Here follows an example.

Load the Seurat package.

```r
library(Seurat, quietly = TRUE)
```

```
##
## Attaching package: 'SeuratObject'

## The following object is masked from 'package:base':
##
##     intersect
```

In the below example, we are going to use a Seurat object called 'pbmc_small' provided by the Seurat package. In real applications, a Seurat project obtained after running the standard Seurat pipeline should be prepared. The Seurat project should have cell clustering available. Use FindAllMarkers() function to generate the differential gene table if you haven't done so:

```
data("pbmc_small")
suppressWarnings({
  all.markers <- FindAllMarkers(object = pbmc_small)
})
```

```
## Calculating cluster 0

## For a (much!) faster implementation of the Wilcoxon Rank Sum Test,
## (default method for FindMarkers) please install the presto package
## --------------------------------------------
## install.packages('devtools')
## devtools::install_github('immunogenomics/presto')
## --------------------------------------------
## After installation of presto, Seurat will automatically use the more
## efficient implementation (no further action necessary).
## This message will be shown once per session

## Calculating cluster 1

## Calculating cluster 2
```

Perform cell type annotation by GPT-4 using the gptcelltype() function. Here you can optionally provide the actual name of the tissue for your dataset.

```
res <- gptcelltype(all.markers,
            tissuename = 'human PBMC',
            model = 'gpt-4'
)
```

```
## [1] "Note: OpenAI API key found: returning the cell type annotations."
## [1] "Note: It is always recommended to check the results returned by GPT-4 in case of\n AI hallucina
```

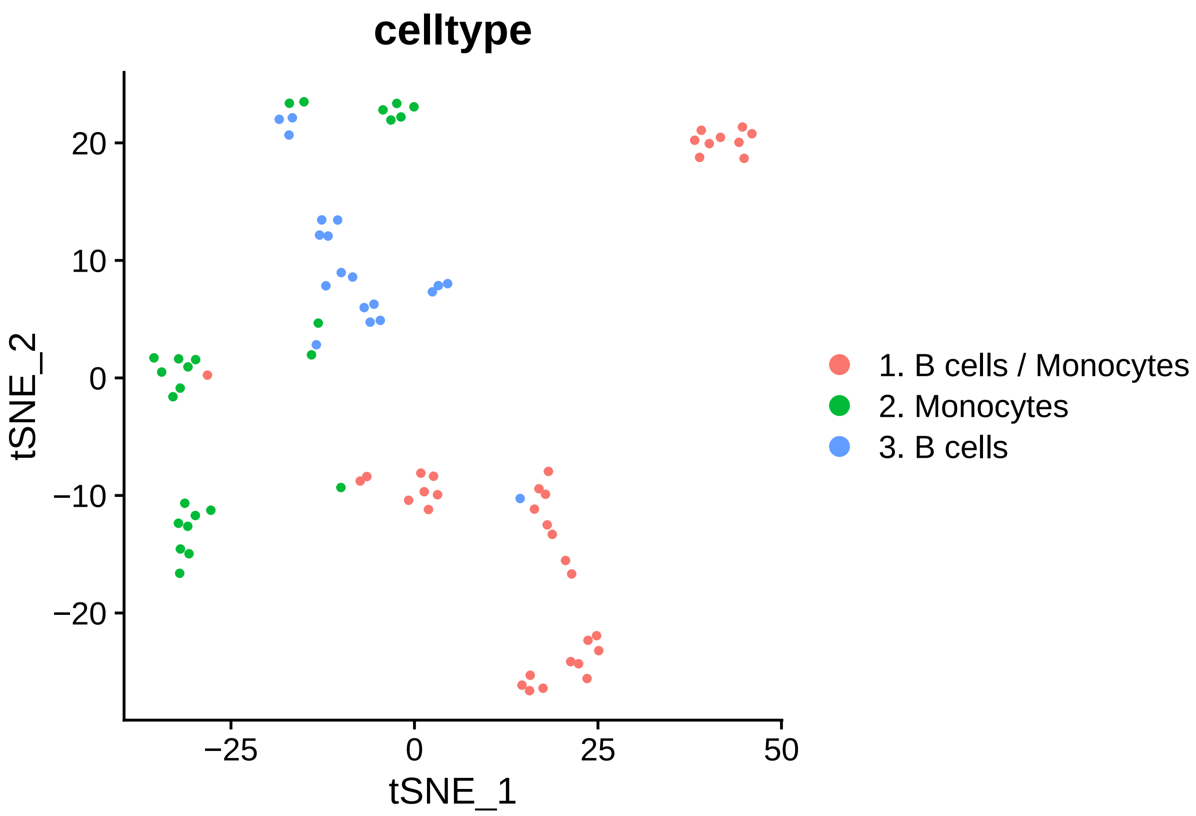
It is always recommended to check the results returned by GPT-4 in case of AI hallucination, before going to down-stream analysis.

```
res
```

```
##                          0                         1                         2
## "1. B cells / Monocytes"           "2. Monocytes"              "3. B cells"
```

If the results make sense, we can assign the cell type annotations back to the Seurat object and visualize the cell type annotations on the UMAP:

```
pbmc_small@meta.data$celltype <- as.factor(res[as.character(Idents(pbmc_small))])
DimPlot(pbmc_small,group.by='celltype')
```



If the results need to be fine-tuned, it is easy to reassign cell type annotations for some clusters. For example, to change the cell type annotation for cluster 0:

```
res[1] <- 'Classical monocytes'
pbmc_small@meta.data$celltype <- res[as.character(Idents(pbmc_small))]
```

If you prefer not to link to GPT-4 API or do not have OpenAI key, you can set `Sys.setenv(OPENAI_API_KEY = '')`. In this case, the gptcelltype() function will print the prompt directly, which can be copied and pasted into the GPT-4 or ChatGPT online user interface to obtain cell type annotations.

```
Sys.setenv(OPENAI_API_KEY = '')
data("pbmc_small")
suppressWarnings({
  all.markers <- FindAllMarkers(object = pbmc_small)
})
```

```
## Calculating cluster 0

## Calculating cluster 1

## Calculating cluster 2
```

```
res <- gptcelltype(all.markers,
            tissuename = 'human PBMC',
            model = 'gpt-4'
)
```

```
## [1] "Note: OpenAI API key not found: returning the prompt itself."
```

```
cat(res)
```

```
## Identify cell types of human PBMC cells using the following markers separately for each
##  row. Only provide the cell type name. Do not show numbers before the name.
##  Some can be a mixture of multiple cell types.
## 0:HLA-DPB1,HLA-DRB1,HLA-DPA1,HLA-DRA,HLA-DRB5,HLA-DQB1,LYZ,TYMP,HLA-DQA1,HLA-DMB
## 1:S100A8,TYMP,S100A9,LYZ,CST3,FCGRT,LST1,AIF1,TYROBP,IFITM3
## 2:HLA-DPB1,MS4A1,HLA-DQB1,HLA-DRB1,HLA-DRA,TCL1A,CD79A,CD79B,HLA-DPA1,HLA-DRB5
```

Example 2: use a list of genes as input Set up your OpenAI API key as a system environment variable before running GPTCelltype.

```
Sys.setenv(OPENAI_API_KEY = 'your_openai_API_key')
```

If we provide a list of two gene vectors: the first vector contains *CD4* and *CD3D*, and the second vector contains *CD14*, then we can call the function in this way:

```
res <- gptcelltype(
  input = list(cluster1 = c('CD4, CD3D'), cluster2 = 'CD14'),
  tissuename = 'human PBMC',
  model = 'gpt-4'
)
```

```
## [1] "Note: OpenAI API key not found: returning the prompt itself."
```

```
res
```

```
## [1] "Identify cell types of human PBMC cells using the following markers separately for each\n row. (
```

```

sessionInfo()
```

Session Info

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] Seurat_5.0.0       SeuratObject_5.0.0 sp_2.1-1           openai_0.4.1
## [5] GPTCelltype_1.0
##
## loaded via a namespace (and not attached):
##   [1] deldir_1.0-9           pbapply_1.7-2          gridExtra_2.3
##   [4] rlang_1.1.1            magrittr_2.0.3         RcppAnnoy_0.0.21
##   [7] spatstat.geom_3.2-7    matrixStats_1.0.0      ggridges_0.5.4
##  [10] compiler_4.3.1         png_0.1-8              vctrs_0.6.3
##  [13] reshape2_1.4.4        stringr_1.5.0          pkgconfig_2.0.3
##  [16] fastmap_1.1.1         ellipsis_0.3.2         labeling_0.4.3
##  [19] utf8_1.2.3            promises_1.2.1         rmarkdown_2.25
##  [22] purrr_1.0.2           xfun_0.40              jsonlite_1.8.7
##  [25] goftest_1.2-3         later_1.3.1            spatstat.utils_3.0-4
##  [28] irlba_2.3.5.1         parallel_4.3.1         cluster_2.1.4
##  [31] R6_2.5.1              ica_1.0-3              stringi_1.7.12
##  [34] RColorBrewer_1.1-3    spatstat.data_3.0-3    reticulate_1.34.0
##  [37] parallelly_1.36.0     lmtest_0.9-40          scattermore_1.2
##  [40] assertthat_0.2.1      Rcpp_1.0.11            knitr_1.44
##  [43] tensor_1.5            future.apply_1.11.0    zoo_1.8-12
##  [46] sctransform_0.4.1     httpuv_1.6.11          Matrix_1.6-1.1
##  [49] splines_4.3.1         igraph_1.5.1           tidyselect_1.2.0
##  [52] abind_1.4-5           yaml_2.3.7             spatstat.random_3.2-1
##  [55] codetools_0.2-19      miniUI_0.1.1.1         spatstat.explore_3.2-5
##  [58] curl_5.1.0            listenv_0.9.0          lattice_0.21-9
##  [61] tibble_3.2.1          plyr_1.8.9             withr_2.5.1
##  [64] shiny_1.7.5           ROCR_1.0-11            evaluate_0.22
##  [67] Rtsne_0.16            future_1.33.0          fastDummies_1.7.3
##  [70] survival_3.5-7        polyclip_1.10-6        fitdistrplus_1.1-11
##  [73] pillar_1.9.0          KernSmooth_2.23-22     plotly_4.10.3
##  [76] generics_0.1.3        RcppHNSW_0.5.0         ggplot2_3.4.3
##  [79] munsell_0.5.0         scales_1.2.1           globals_0.16.2
##  [82] xtable_1.8-4          glue_1.6.2             lazyeval_0.2.2
##  [85] tools_4.3.1           data.table_1.14.8      RSpectra_0.16-1
##  [88] RANN_2.6.1            leiden_0.4.3           dotCall64_1.1-0
##  [91] cowplot_1.1.2         grid_4.3.1             tidyr_1.3.0
##  [94] colorspace_2.1-0      nlme_3.1-163           patchwork_1.1.3
##  [97] cli_3.6.1             spatstat.sparse_3.0-3  spam_2.10-0
## [100] fansi_1.0.5           viridisLite_0.4.2      dplyr_1.1.3
## [103] uwot_0.1.16           gtable_0.3.4           digest_0.6.33
## [106] progressr_0.14.0      ggrepel_0.9.4          farver_2.1.1
## [109] htmlwidgets_1.6.2     htmltools_0.5.6.1      lifecycle_1.0.3
## [112] httr_1.4.7            mime_0.12              MASS_7.3-60
```