Detecting Duplicated Code

Software Evolution and Reverse Engineering

Refolli F. 865955

March 2, 2025

Why?

The magic of RPM Macros

```
AutoReq:        no
Name:           packer
Version:        1.0.0
Release:        1
BuildArch:      x86_64
Summary:        Yet another rpmbuild/makepkg wrapper
License:        GPL-3.0
Source:         packer.tar.gz
BuildRequires:  meson make yaml-cpp-devel
Requires:       yaml-cpp

Url: https://github.com/frefolli/packer
Prefix: /usr

%description
Yet another rpmbuild/makepkg wrapper

%prep
%setup -n packer-master

%install
# reset buildroot
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT
%make_build
%make_install
find $RPM_BUILD_ROOT -type f,l | sed "s|^$RPM_BUILD_ROOT|/|" > .extra_files

%clean
rm -rf $RPM_BUILD_ROOT

%files -f .extra_files
```

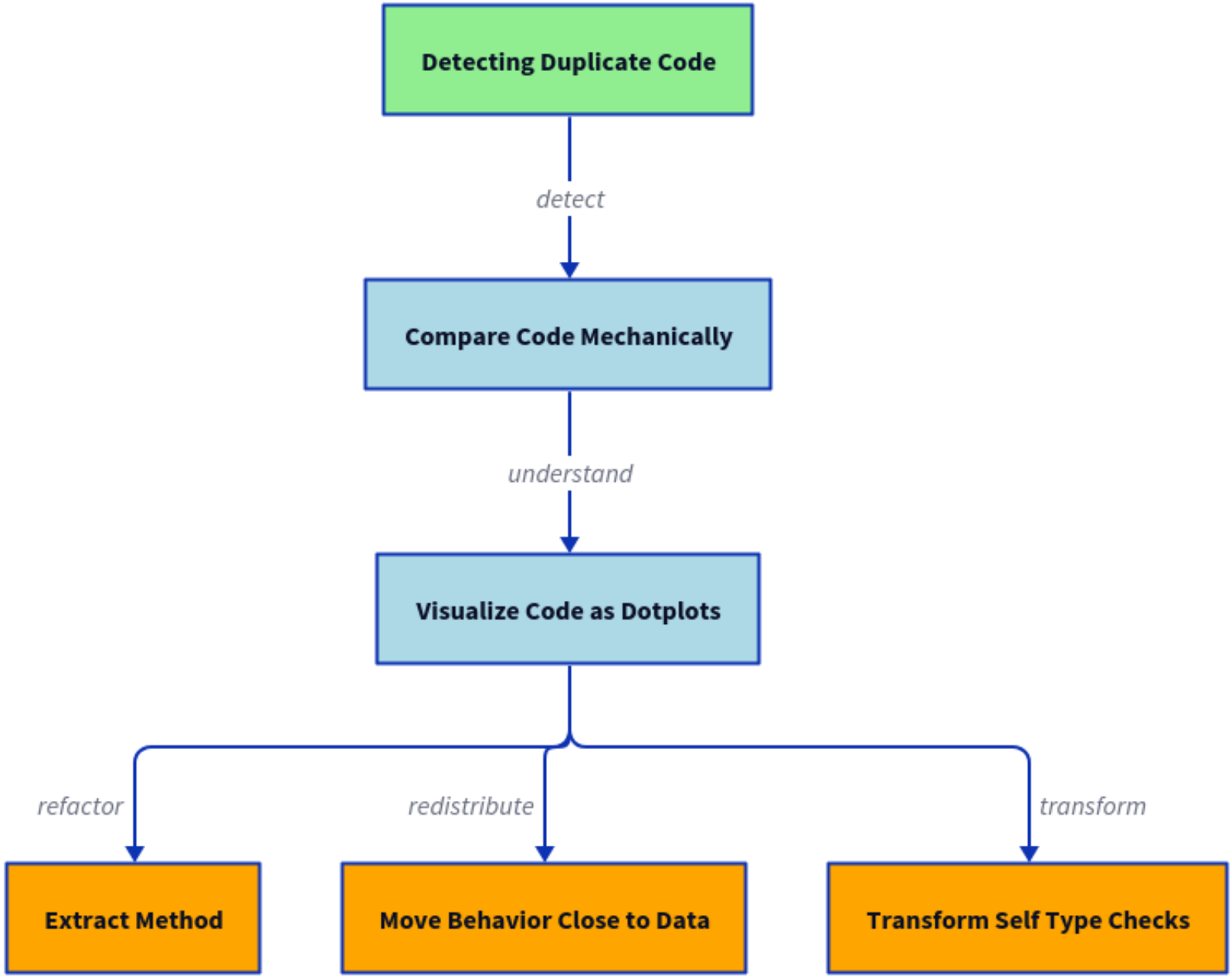
make_install macro

```
#-----------------------------------------------------------------------
# The make install analogue of %configure for modern autotools:
%make_install %{__make} install DESTDIR=%{?buildroot} INSTALL="%{__install} -p"
```

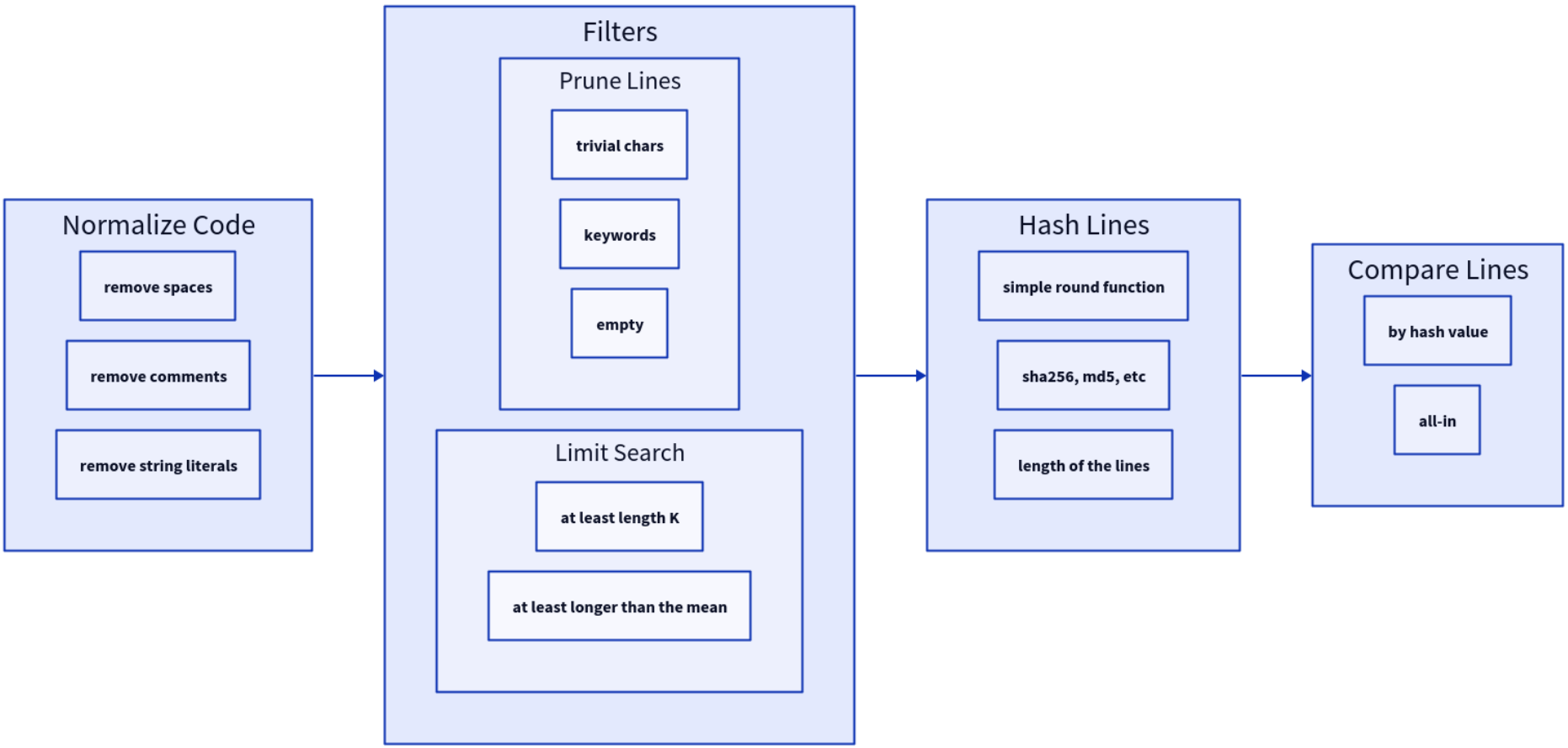
makeinstall macro

```
#-----------------------------------------------------------------------
# Former make install analogue, kept for compatibility and for old/broken
#  packages that don't support DESTDIR properly.
%makeinstall \
  echo "warning: %%makeinstall is deprecated, try %%make_install instead" 1>&2\
  %{__make} \\\
  prefix=%{?buildroot:%{buildroot}}%{_prefix} \\\
  exec_prefix=%{?buildroot:%{buildroot}}%{_exec_prefix} \\\
  bindir=%{?buildroot:%{buildroot}}%{_bindir} \\\
  sbindir=%{?buildroot:%{buildroot}}%{_sbindir} \\\
  sysconfdir=%{?buildroot:%{buildroot}}%{_sysconfdir} \\\
  datadir=%{?buildroot:%{buildroot}}%{_datadir} \\\
  includedir=%{?buildroot:%{buildroot}}%{_includedir} \\\
  libdir=%{?buildroot:%{buildroot}}%{_libdir} \\\
  libexecdir=%{?buildroot:%{buildroot}}%{_libexecdir} \\\
  localstatedir=%{?buildroot:%{buildroot}}%{_localstatedir} \\\
  sharedstatedir=%{?buildroot:%{buildroot}}%{_sharedstatedir} \\\
  mandir=%{?buildroot:%{buildroot}}%{_mandir} \\\
  infodir=%{?buildroot:%{buildroot}}%{_infodir} \\\
  install
```

Overview of the Cluster



Compare Code Mechanically



Example of Normalization

```python
def parse_file_range(inp: str) -> FileRange:
  splitted_inp: list[str] = inp.strip().split(':')
  if len(splitted_inp) == 1:
    path = splitted_inp[0]
    path = os.path.normpath(os.path.expanduser(path))
    if not os.path.exists(path):
      raise ValueError(inp)
    return (path, None, None)
  elif len(splitted_inp) == 3:
    path = splitted_inp[0]
    path = os.path.normpath(os.path.expanduser(path))
    if not os.path.exists(path):
      raise ValueError(inp)
    return (path, int(splitted_inp[1]), int(splitted_inp[2]))
  raise ValueError(inp)
```

Figure: Before

```python
defparse_file_range(inp:str)->FileRange:
splitted_inp:list[str]=inp.strip().split(':')
iflen(splitted_inp)==1:
path=splitted_inp[0]
path=os.path.normpath(os.path.expanduser(path))
ifnotos.path.exists(path):
raiseValueError(inp)
return(path,None,None)
eliflen(splitted_inp)==3:
path=splitted_inp[0]
path=os.path.normpath(os.path.expanduser(path))
ifnotos.path.exists(path):
raiseValueError(inp)
return(path,int(splitted_inp[1]),int(splitted_inp[2]))
raiseValueError(inp)
```

Figure: After

Dupdec: a duplicate code detection tool

```
usage: __main__.py [-h] [-l LANG] [-M] [-D] [-o OUTPUT] [-f {json,yaml}] [-t TOL] FILE [FILE ...]

detect duplicates code via Mechanical Code Comparison

positional arguments:
  FILE                  files to analyze, in <PATH>[:STARTLINE:ENDLINE] format

options:
  -h, --help            show this help message and exit
  -l LANG, --lang LANG  specifies the programming languages used in files, if not issued it's automatically inferred from input names
  -M, --mechanical      mechanical comparison
  -D, --dotplot         dotplot production
  -o OUTPUT, --output OUTPUT
                        where to store output (file for mechanical | dir for dotplot)
  -f {json,yaml}, --format {json,yaml}
                        how to store mechanical output
  -t TOL, --tol TOL     edit distance tolerance for line matching
```

Dupdec: self-audit

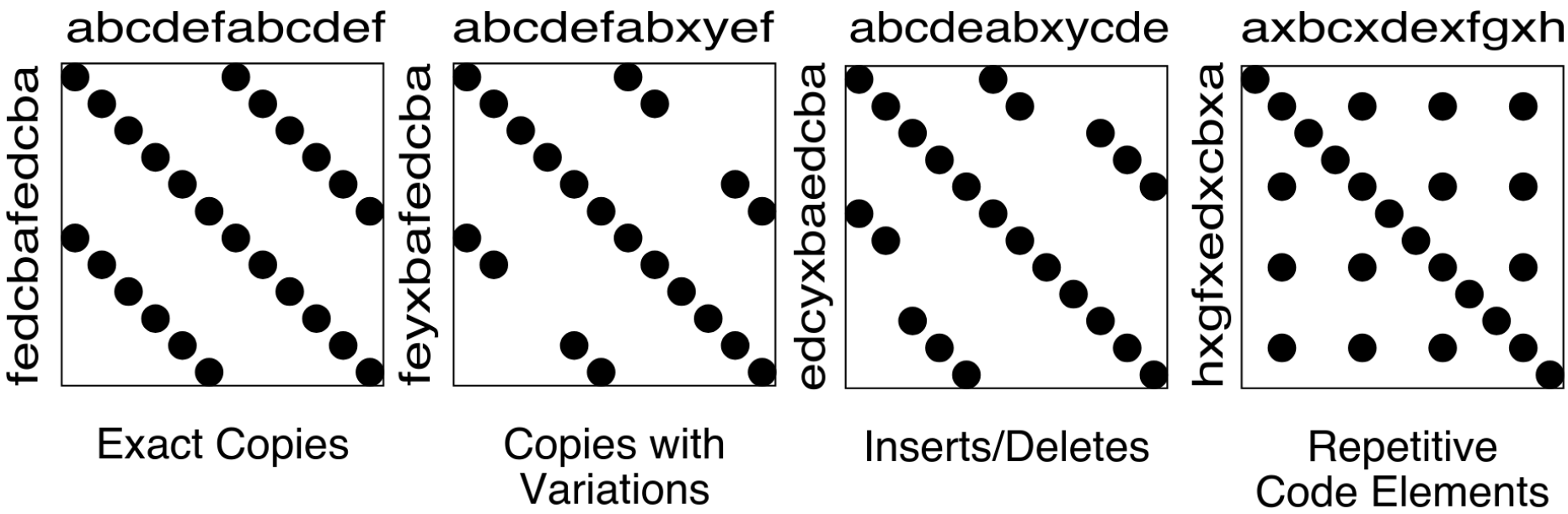
dupdec/main.py:0:329 | dupdec/main.py:0:329

LFile	LLine	<>	RLine	RFile
dupdec/**main**.py:15	`IOTA_COUNTER=0`	<>	`IOTA_COUNTER = 0`	dupdec/**main**.py:19
dupdec/**main**.py:18	`global IOTA_COUNTER`	<>	`global IOTA_COUNTER`	dupdec/**main**.py:22
dupdec/**main**.py:19	`IOTA_COUNTER = 0`	<>	`IOTA_COUNTER=0`	dupdec/**main**.py:15
dupdec/**main**.py:22	`global IOTA_COUNTER`	<>	`global IOTA_COUNTER`	dupdec/**main**.py:18
dupdec/**main**.py:38	`repr += ":%d" % file_range[1]`	<>	`repr += ":%d" % file_range[2]`	dupdec/**main**.py:39
dupdec/**main**.py:39	`repr += ":%d" % file_range[2]`	<>	`repr += ":%d" % file_range[1]`	dupdec/**main**.py:38
dupdec/**main**.py:67	`return lang`	<>	`return lang`	dupdec/**main**.py:85
dupdec/**main**.py:68	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:93
dupdec/**main**.py:68	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:99
dupdec/**main**.py:68	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:101
dupdec/**main**.py:76	`if lang is None:`	<>	`if lang is None:`	dupdec/**main**.py:83
dupdec/**main**.py:83	`if lang is None:`	<>	`if lang is None:`	dupdec/**main**.py:76
dupdec/**main**.py:85	`return lang`	<>	`return lang`	dupdec/**main**.py:67
dupdec/**main**.py:90	`path = splitted_inp[0]`	<>	`path = splitted_inp[0]`	dupdec/**main**.py:96
dupdec/**main**.py:91	`path = os.path.normpath(os.path.expanduser(path))`	<>	`path = os.path.normpath(os.path.expanduser(path))`	dupdec/**main**.py:97
dupdec/**main**.py:92	`if not os.path.exists(path):`	<>	`if not os.path.exists(path):`	dupdec/**main**.py:98
dupdec/**main**.py:93	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:68
dupdec/**main**.py:93	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:99
dupdec/**main**.py:93	`raise ValueError(inp)`	<>	`raise ValueError(inp)`	dupdec/**main**.py:101

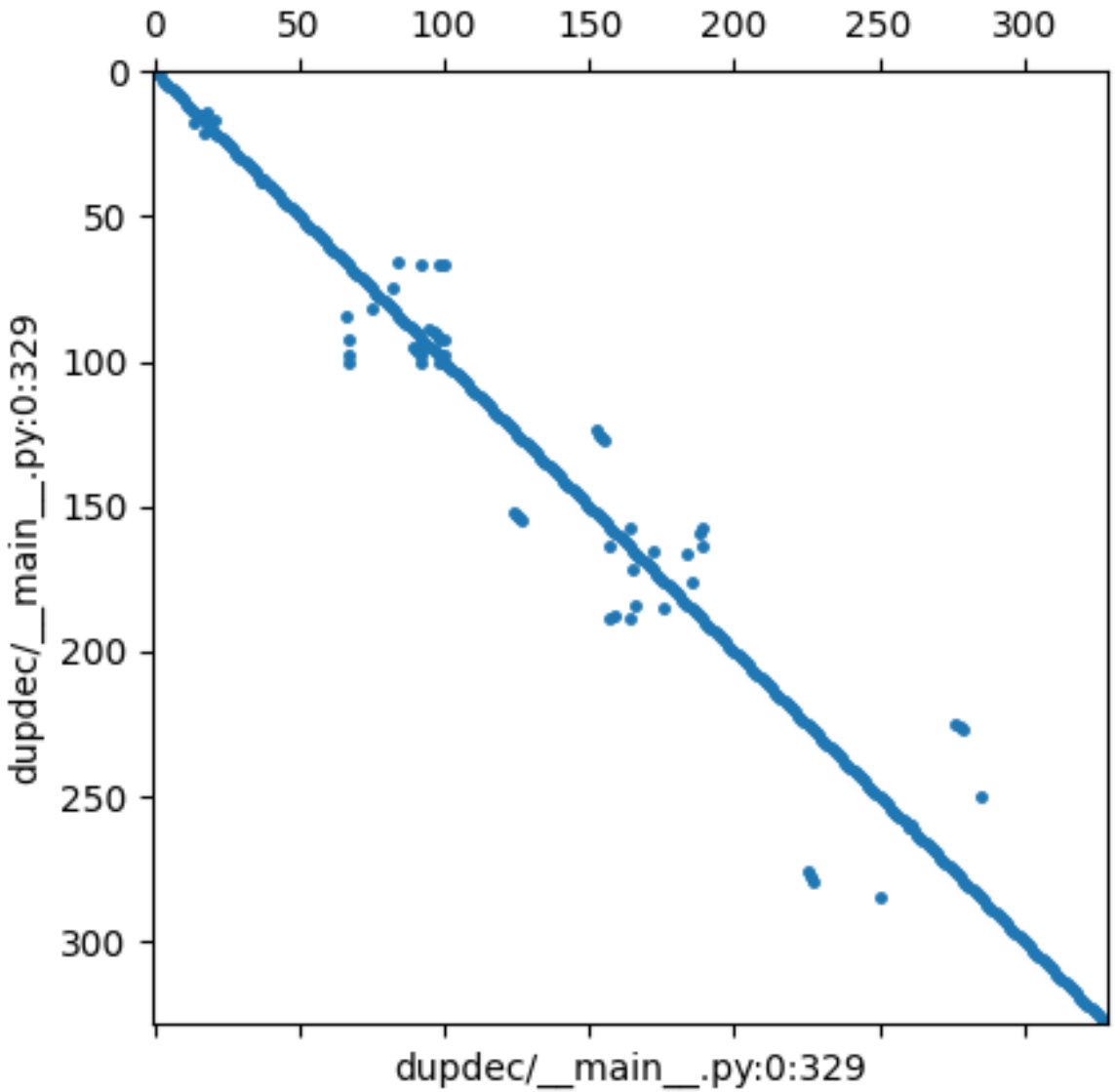
What has dupdec found?

```python
def parse_file_range(inp: str) -> FileRange:
  splitted_inp: list[str] = inp.strip().split(':')
  if len(splitted_inp) == 1:
    path = splitted_inp[0]
    path = os.path.normpath(os.path.expanduser(path))
    if not os.path.exists(path):
      raise ValueError(inp)
    return (path, None, None)
  elif len(splitted_inp) == 3:
    path = splitted_inp[0]
    path = os.path.normpath(os.path.expanduser(path))
    if not os.path.exists(path):
      raise ValueError(inp)
    return (path, int(splitted_inp[1]), int(splitted_inp[2]))
  raise ValueError(inp)
```

Dotplot almanac



Dupdec: self-dotplot



Tradeoffs

- Pros
  - Simple process
  - Mostly language independent
  - Graphical visualizations ease first approach to unknown code
- Cons
  - Scaling issues, optimizations required (sparse matrices, smart filters ... etc)
  - Possible information overload
  - Dotplots can be misleading if not followed by an in depth code exploration

The End