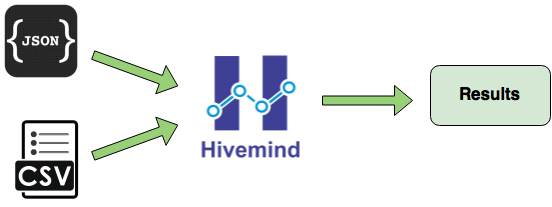
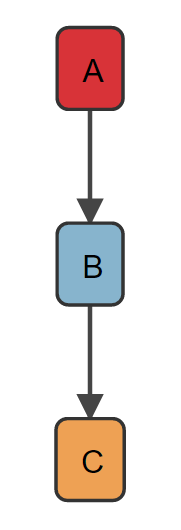
User Guide

This section serves as a user guide for the Hivemind package version 0.1.0. Hivemind is designed to allow users to build a variety of medical pipelines and run them across a distributed cluster for large number of patients to speed up overall computation time.

**Getting Started**



**Figure 1**



Here is an example of shell script running a three stage pipeline as showing in Figure 2. The script will run each stage in the pipeline sequentially.

#!/bin/sh

/bin/task-1 -o /cat /dog

/bin/task-2 -g -i /cat

/bin/task-3 /cat

**Figure 2**

Now suppose that after task-A has completed, both task-B and task-C can be run. This cannot be easily expressed in a sequential shell script. Hivemind can handle this situation with ease.

To start building a pipeline, Hivemind uses a JSON specification for representing the structure of a pipeline. The user needs to provide the structure of the pipeline in a JSON file with the specification shown below. The user also needs to provide patients data in the form of a CSV file.

\_uid : a unique task id

args : list of arguments passed

exe : the executable that needs to be run

wd : working directory for the executable

\_requires : list of dependencies for this task

verify\_exe : post-stage verification executable; exe and args will also be passed to this

skip : whether to skip this task or not (useful when certain patients do not require certain tasks)

shell : whether the task need to run using the shell or not (useful when you need shell features such as redirecting output)

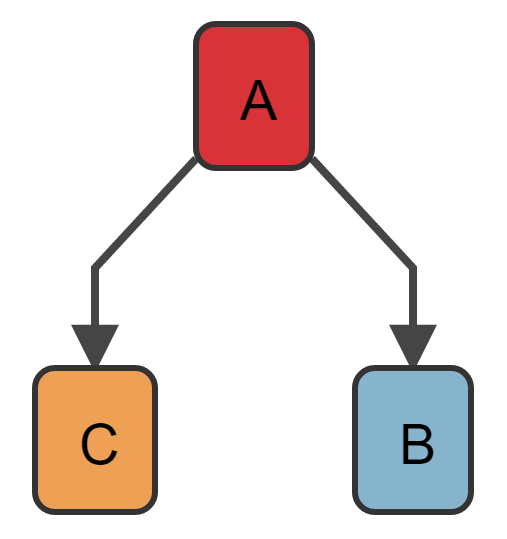
Here is an example of the same pipeline built using said JSON specification and a CSV file containing three patients. In the Hivemind version of the pipeline, it is possible for tasks B and C to run at the same time. Note that the $$ variables in the JSON file will be replaced by the values in the CSV files for each patient (further details below).  
[

{

"\_uid": "A",

"args": ["-o", "/$$arg$$", "/$$arg2$$"],

"exe": "bin/task-1",

"wd": "/workingDirectory"

},

{

"\_uid": "B",

"args": ["-g", "-i", "/$$arg$$"],

"exe": "/bin/task-2",

"wd": "/workingDirectory",

"\_requires": ["A"],

**Figure 3**

"verify\_exe": "/bin/verify"

},

{

"\_uid": "C",

"\_requires": ["A"],

"args": ["/$$arg$$"],

"exe": "/bin/task-3",

"wd": "/workingDirectory",

"verify\_exe": "/bin/verify"

}

]

As mentioned above, fields with $$ at each end (such as $$arg$$) represent a variable specified in the CSV file which will be replaced for each patient. The example below shows a simple CSV file which includes three patients’ data. Columns that do not contain variables are ignored.

patientID,$$arg$$,$$arg2$$

123,cat,dog

456,fish,gerbil

789,snake,bird

Hivemind will combine the JSON and CSV files to create a specific pipeline for each patient that can be executed in a distributed cluster as showing in Figure 1.

**Usage**

After specifying the structure of the pipeline in a JSON file and the patients data in a CSV file, the user can start Hivemind using the mpirun command for executing the pipeline.

The following command will run MPI using the mpirun command with 5 processes (one master and four workers). python main.py will run the the actual pipeline.

mpirun -n 5 python main.py -j pipeline.json -c patients.csv

The detailed usage of main.py is below.

usage: main.py [-h] -j JSON [JSON ...] -c CSV [CSV ...] [-p CHECKPOINT]

[-r {0,1}] [-d]

Example of running a MPI pipeline

optional arguments:

-h, --help show this help message and exit

-j JSON [JSON ...], --json JSON [JSON ...]

JSON files

-c CSV [CSV ...], --csv CSV [CSV ...]

CSV files

-p CHECKPOINT, --checkpoint CHECKPOINT

Optional checkpoint directory

-r {0,1}, --ranker {0,1}

Optional rank function ID as int (for task priority)

0: rank\_by\_total\_successors, 1: rank\_by\_successors

-d, --dry-run Only print commands rather than run them

-j : passing one or multiple JSON files

-c : passing one or multiple CSV files

-p : checkpoint directory (if unspecified, Hivemind will create a directory in /tmp/hivemind for the pipeline based on the current date and time)

-r : whether to rank the tasks by total successors or immediate successors (default is FIFO)

-d : printing only the commands without running them (useful for testing, this turns off checkpointing since no work is performed)

The following command specifies multiple JSON files and a directory for checkpointing

mpirun -n 5 python main.py -j pipeline.json pipeline2.json -c patients.csv -p /home/cpDir

The following command specifies the ranking method of the tasks to run in immediate successors order

mpirun -n 5 python main.py -j pipeline.json -c patients.csv -r 1

The following example prints out all the commands that would be run so that the user can verify the pipeline

mpirun -n 5 python -O main.py -j pipeline.json -c patients.csv -d

The output (colored per patient) of the above JSON and CSV file would be (note that Hivemind’s debug messages were suppressed using python’s -O flag):

/bin/task-1 -o /cat /dog

/bin/task-1 -o /snake /bird

/bin/task-3 /cat

/bin/task-3 /snake

/bin/task-2 -g -i /cat

/bin/task-2 -g -i /snake

/bin/task-1 -o /fish /gerbil

/bin/task-2 -g -i /fish

/bin/task-3 /fish

Hivemind allows the user to generate a visual representation of the pipeline to verify the sequence of the stages and the dependencies within the pipeline. The following command will run d3.py and open a new browser window showing a graphical representation of the given JSON file(s).

python d3.py -j pipeline-1.json pipeline-2.json -s 1000 -t 80

The detailed usage of d3.py is below.

usage: d3.py [-h] -j JSON [JSON ...] [-s SEED] [-t CUTOFF]

Utility for 'printing' a pipeline

optional arguments:

-h, --help show this help message and exit

-j JSON [JSON ...], --json JSON [JSON ...]

JSON files

-s SEED, --seed SEED Optional seed for random coloring

-t CUTOFF, --cutoff CUTOFF

Cutoff value for fuzzy string matching

-j : Passing one or multiple JSON files

-s : optional seed for coloring the tasks within the pipeline (default is 2016)

-t : optional fuzzy string cutoff for color matching executables and arguments (default is 90)