VELD

Versioned Executable Logic and Data

Stefan Resch

stefan.resch@oeaw.ac.at

Austrian Centre for Digital Humanities and Cultural Heritage

Technical Concept: https://doi.org/10.5281/zenodo.13322913

Formal Metadata Specification: https://github.com/acdh-oeaw/VELD_spec

Current collection of VELD repos: https://github.com/veldhub/

Demo: https://github.com/veldhub/veld_chain__demo_wordembeddings_multiarch

VELD: outline

- CONTEXT
- DEMO (execution)
- IMPLEMENTATION: DESIGN PATTERN
- IMPLEMENTATION: METADATA SCHEMA
- DEMO (details)
- SUMMARY
- ROADMAP
- QUESTIONS / FEEDBACK

CONTEXT

What is VELD?

- A design pattern, with a reference implementation based solely on:
  - Git
  - Docker
- A metadata schema
- Originated in the CLS (Computational Literature Studies) Infra project

What problems are addressed by CLS Infra and VELD?

Strong overlap between Computational Literature Studies and Data Science. Therein, recurring pain points are:

Fragile Reproducibility of workflows

- Hidden dependencies
- Sprawling complexity
- Changing environments

High Integration costs / low reusability of implemented code

- Strong coupling of code and data lowers reuse across potential contexts
- Hardwired assumptions
- Leaky abstractions
- Poor interface designs

What are the goals of CLS Infra / VELD?

- Encapsulating CLS / NLP tools to ease their usage, especially by less tech-savvy scholars
- Increase reuse, adaptability and interoperability of produced modules
- Make data transformation paths, entire workflows, reproducible with as little friction as possible
- Focus on small to medium local workflows / tools

DEMO (execution)

shorter.me/94YkN

https://github.com/veldhub/veld_chain__demo_wordembeddings_multiarch

IMPLEMENTATION:

Design Pattern

What is VELD's guiding architectural philosophy?

Robert C. Martin's excellent architectural principles and how we implement them:

- **Hiding implementation details**
  -> Encapsulation of arbitrary software complexity within docker container
- **Providing a clear interface**
  -> Defining a consistent metadata schema across tools
- **Single responsibility principle**
  -> Focus on atomic task-driven modules
- **Composition over inheritance**
  -> modules can be aggregated

Design Pattern

As a **design pattern**, VELD can be implemented in different ways and freely adapted, and as a **reference implementation**, it is based solely on:

- Git (submodules)
- Docker (compose)

Three kinds of git repositories representing VELD objects (called "veld"):

- **data veld:** atomic unit containing static serialized data
- **code veld:** atomic unit containing source code and docker compose files
- **chain veld:** aggregation of data and code velds

data veld

The data veld represents stand-alone data, and may also be used as input or output for code or chain velds.

- Contains static serialized data
- Should also contain structured metadata, describing itself:
  - What file type? (e.g. txt)
  - What topics? (e.g. NLP)
  - Description? (e.g. its origin)

code veld

The code veld represents a generic atomic unit of computation. Its implementation is abstracted away with docker, and it can be executed either on its own or integrated into a chain veld.

- Contains source code and docker compose files
- Compose file also contains non-docker metadata, describing itself:
  - What file types does it take as input / output? (e.g. input: txt & output: conllu)
  - Where inside the container does it expect input and output? (e.g. `/veld/input/`)
  - What parameters does it expect? (e.g. hyperparameters for NLP training)
- Docker compose file acts as **single execution point of code**

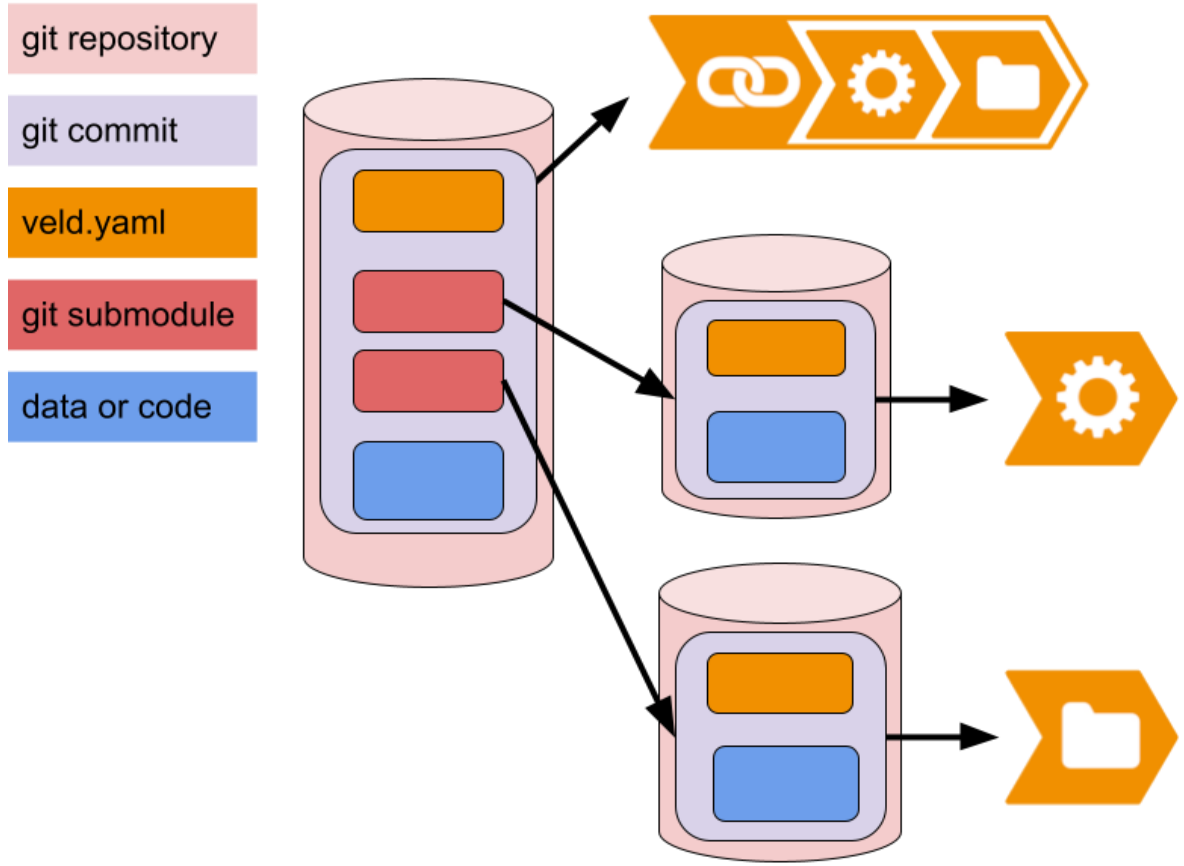
chain veld

The chain veld acts as the aggregation of specific code and data. It persists the entire context of a workflow and makes it reproducible with one execution point.

- Contains data and code velds, aggregated as git submodules (or non-veld data / code)
- Also contains docker compose files, but those **extend the code velds**:
  - Reuses functionality from a code veld
  - Can override volume mounts, which is the main way of loading data into code velds
  - Can override configuration to adapt functionality to a given workflow

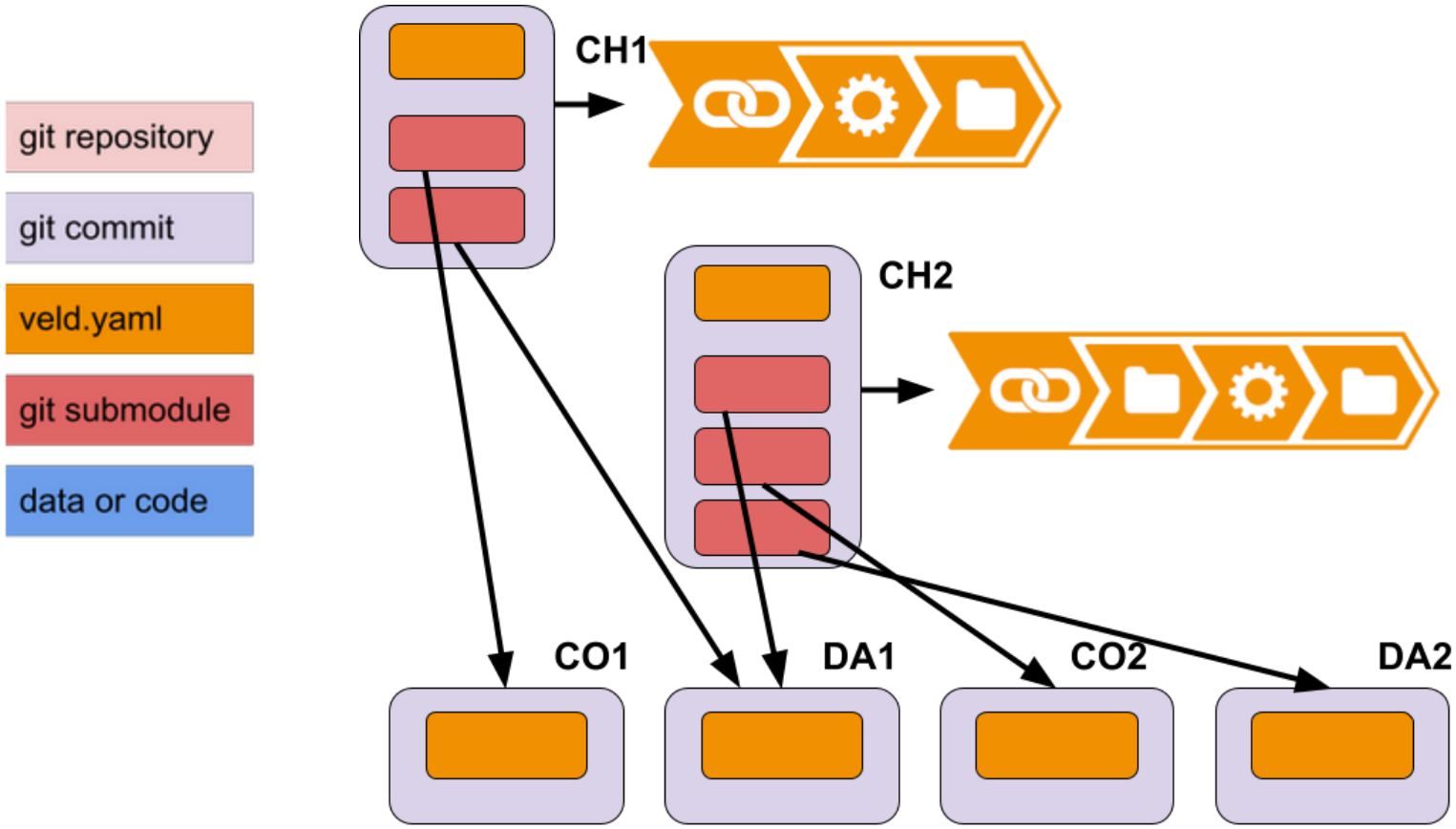
Composition: submodules

A chain is a **composition of its modules** via git submodules



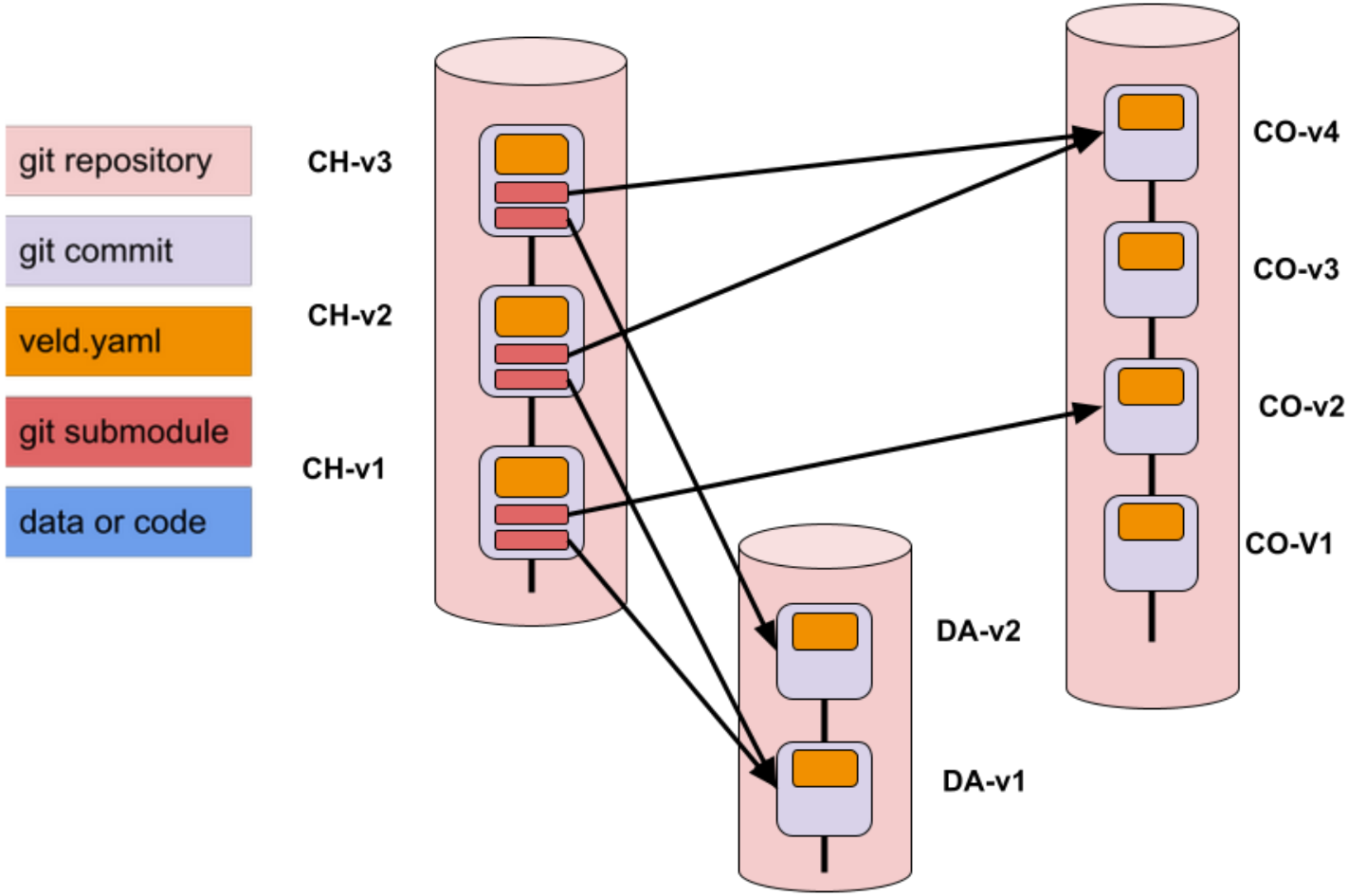
Flexibility: modular data and code units

By splitting workflows into these three types, **reuse of the atomic modules** (data and code) **across different contexts** can be increased



Stability: superstate chain and its history

Since a chain integrates data and code velds with git submodules, their respective states (expressed as commits) are **aggregated into a cohesive superstate** with full history of itself and its modules.



IMPLEMENTATION:

Metadata Schema

Metadata schema: across all velds

- A veld object is described with a veld yaml file (named either **veld.yaml** or **veld_<SOME_NAME>.yaml**)
- Every veld yaml file must contain a section

x-veld:

<VELD_TYPE>:

<METADATA>

Metadata schema: data veld

A data veld yaml only contains **metadata**.

```
x-veld:    # metdata
  data:
    file_type: json
```

Metadata schema: code veld

A code veld yaml is a **docker compose yaml file**, that contains **non-docker metadata** and a single **docker compose service**.

```
x-veld:    # metdata
 code:
  input:
   volume: /veld/input/
   file_type: json

services:    # docker compose service
 veld_code:
  build: .
  command: python /veld/code/run.py
```

Metadata schema: chain veld

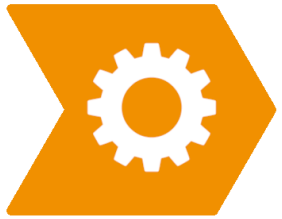
A chain veld yaml is a **docker compose yaml file**, that contains **metadata** and one or more **docker compose services** which may **extend** a service from a **code veld**.

```
x-veld:    # metdata
 chain:


services:    # docker compose service
 veld_chain:
  extends:    # extends a code veld
   file: ./code_veld_sub_repo/veld.yaml
   service: veld_code
  volumes:
   - ./data_veld_sub_repo/:/veld/input/    # mounting a data veld
```

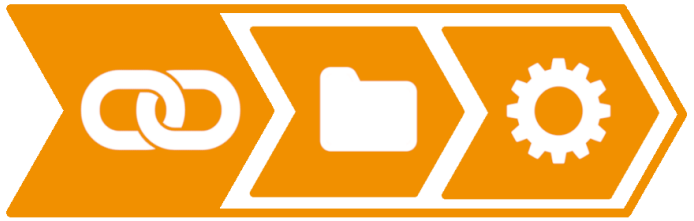
Metadata schema: matching aggregation example

```
x-veld:
  data:
    file_type: json
```



```
x-veld:
  code:
    input:
      volume: /veld/input/
      file_type: json

services:
  veld_code:
    build: .
    command: python /veld/code/run.py
```

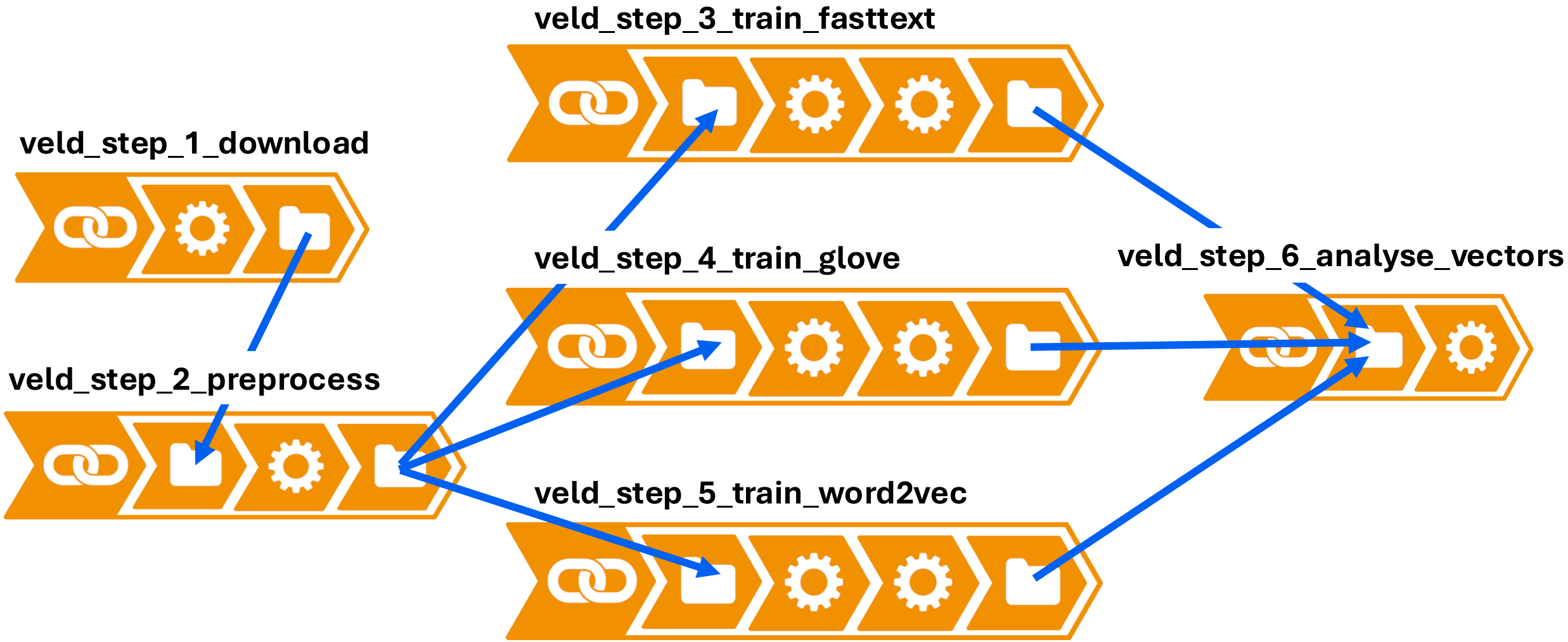


```
x-veld:
  chain:


services:
  veld_chain:
    extends:
      file: ./code_veld_repo/veld.yaml
      service: veld_code
    volumes:
      - ./data_veld_repo/:/veld/input/
```

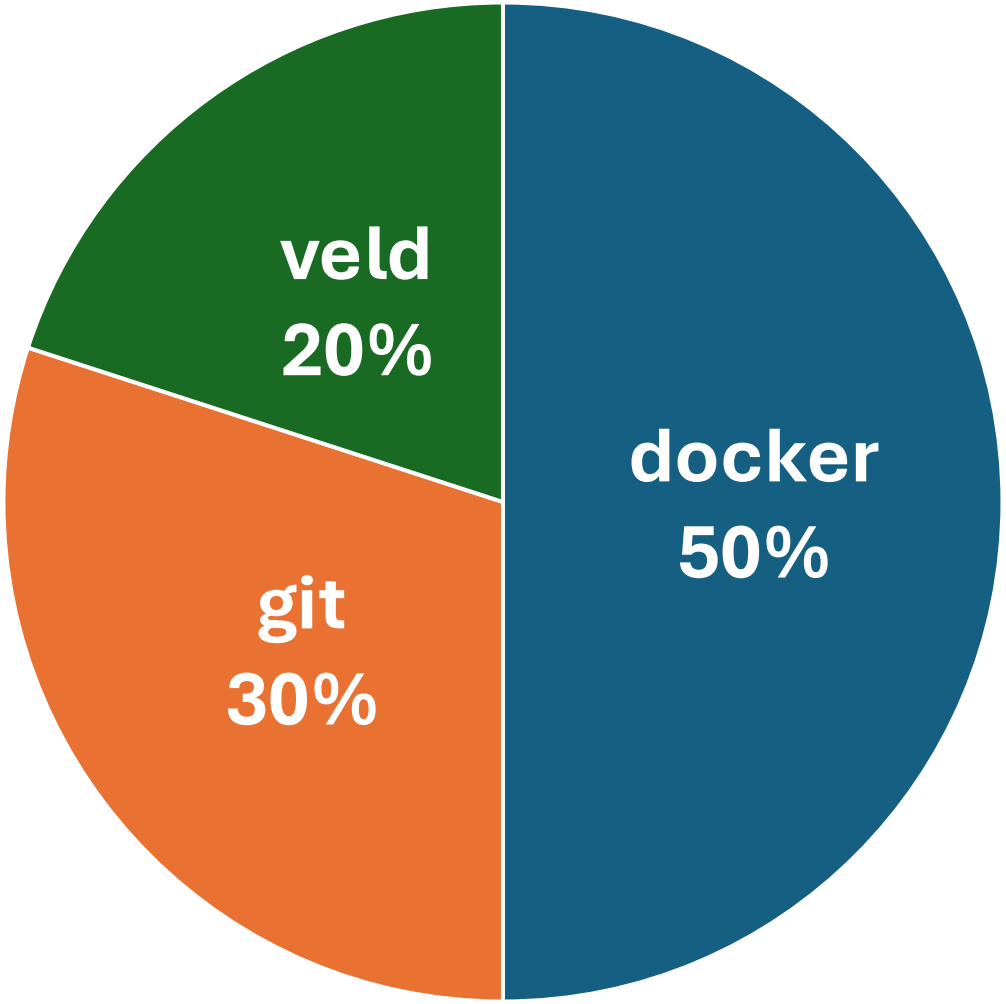
DEMO (details)

Demo: overview of velds



SUMMARY

Necessary skill set for VELD (rough estimation)



Suitable contexts for VELD

More suitable in a **"wide"** context ("many small things")

- Workflows that can be executed on local hardware
- Hetereogenous organization
- Decentralized infrastructure
- High fluctuation of code stacks and employees / collaborators

Less suitable in a **"tall"** context ("few big things")

- Workflows that inherently require cloud services or enormous hardware
- In a homogenous organization
- Centralized infrastructure
- Low fluctuation of code stacks and employees / collaborators

Weighing advantages and disadvantages

Advantages

- **Both high flexibility and stability** (usually contrary characteristics!)
- Docker is (in my biased experience) **the best tool** for encapsulating complexity in **an economic way**
- Isolating complexity inherently drives focus on interface
- Based on a widely used and mature stack, carry-over effect to other tasks
- Increased security through sandboxing
- Flexible opt-in design

Disadvantages

- **"raw" design pattern**, with an inherent learning curve regarding the tools
- Git submodules are a bit tricky
- Some inherent conflicts between docker compose syntax and VELD pragmatics
- No VELD client which would help with integration

ROADMAP

Roadmap

- CLS Infra is coming to an end, where we have implemented a reasonable amount of VELD objects
- We will extend usage of VELD internally at our institute, applying it at other Data Science projects
- In parallel, we look into creating a platform (maybe called "veldhub") where VELD metadata is aggregated to facilitate discovery and interoperability of VELD objects and projects
- We also welcome guinea pigs *wink wink*

THANK YOU!

QUESTIONS / FEEDBACK?