# A security analyst's guide to selecting training data

Luke Dramko

We build machine-learning based tools to help security researchers.

# We build machine-learning based tools to help security researchers.

DIRE `int v1` ⟶ `int size`

# We build machine-learning based tools to help security researchers.

DIRE `int v1` ⟶ `int size`

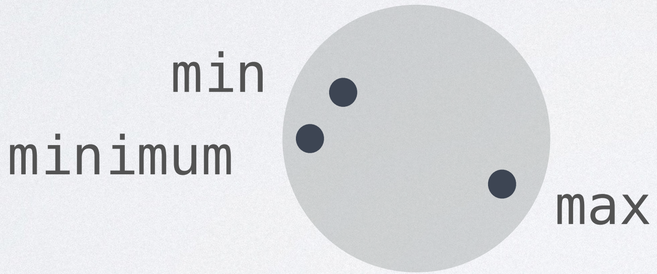
DIRTY `_QWORD *` ⟶ `struct point *`

# We build machine-learning based tools to help security researchers.

DIRE `int v1` ⟶ `int size`

DIRTY `_QWORD *` ⟶ `struct point *`

VarCLR

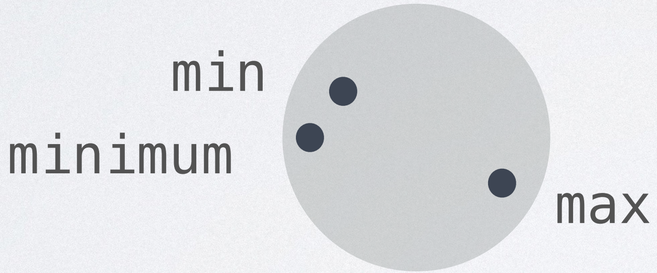


# We build machine-learning based tools to help security researchers.

DIRE `int v1` ⟶ `int size`

DIRTY `_QWORD *` ⟶ `struct point *`

VarCLR



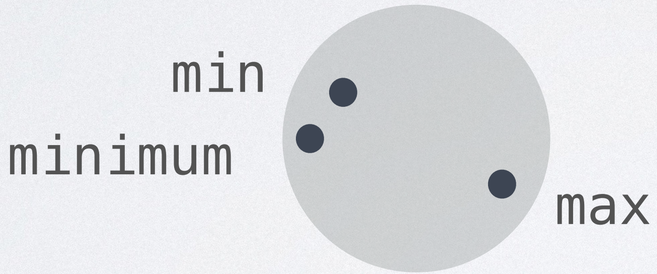

SILO

# We build machine-learning based tools to help security researchers.

DIRE `int v1` ⟶ `int size`

DIRTY `_QWORD *` ⟶ `struct point *`

VarCLR




SILO


Idioms *(coming soon!)* `*(_QWORD *) (a1 + 4120)` ⟶ `ctrl->used_size`

We studied how the ***characteristics of training data*** influence these machine learning models.

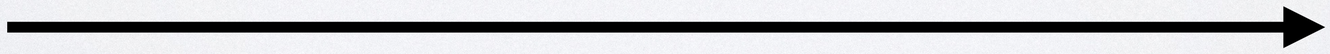
# We use DIRE in our case study.

```
void *file_mmap(int v1, int v2) {
    void * v3;
    v3 = mmap(0, v2, 1, 2, v1, 0);
    if (v3 == (void *) -1) {
        perror("mmap")
        exit(1)
    }
    return v3;
}
```

DIRE

```
void *file_mmap(int fd, int size) {
    void * buf;
    buf = mmap(0, size, 1, 2, fd, 0);
    if (buf == (void *) -1) {
        perror("mmap")
        exit(1)
    }
    return buf;
}
```

v1
v2
v3



fd
size
buf

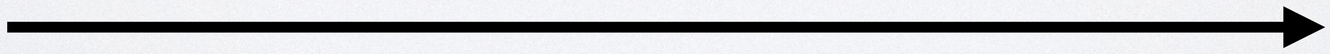
# DIRE is a machine-learning based tool which automatically suggests variable names in decompiled code.

```
void *file_mmap(int v1, int v2) {
    void * v3;
    v3 = mmap(0, v2, 1, 2, v1, 0);
    if (v3 == (void *) -1) {
        perror("mmap")
        exit(1)
    }
    return v3;
}
```

DIRE

```
void *file_mmap(int fd, int size) {
    void * buf;
    buf = mmap(0, size, 1, 2, fd, 0);
    if (buf == (void *) -1) {
        perror("mmap")
        exit(1)
    }
    return buf;
}
```

v1
v2
v3



fd
size
buf

# DIRE is trained on large amounts of code from **GitHub**.

Which **C** programs do we select?

The composition of the training set is important.

# The composition of the training set is important.

# The composition of the training set is important.

# The composition of the training set is important.

# The composition of the training set is important.

# The composition of the training set is important.

# The composition of the training set is important.

# We looked at ways to select and augment data to improve performance, especially on rare and underrepresented variable names.

## Domains

Helping DIRE perform better on domain-specific identifiers.

networking

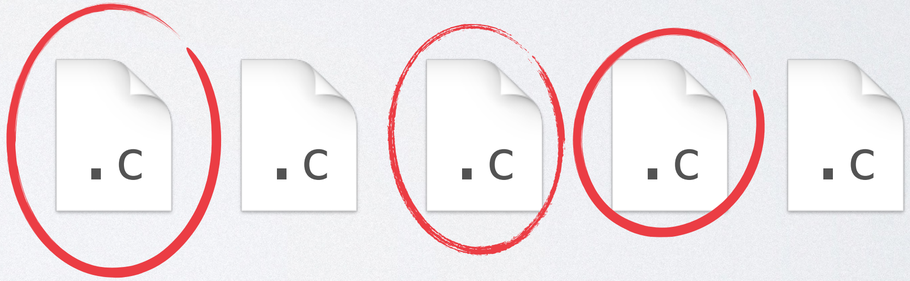
```
http_response
socket          +acc
...
```

compiler

```
lexeme
ast             +acc
...
```

## Dataset Quality

Carefully selecting training data to improve generalizability.



We looked at ways to select and augment data to improve performance, especially on rare and underrepresented variable names.

## Domains

Helping DIRE perform better on domain-specific identifiers.

networking

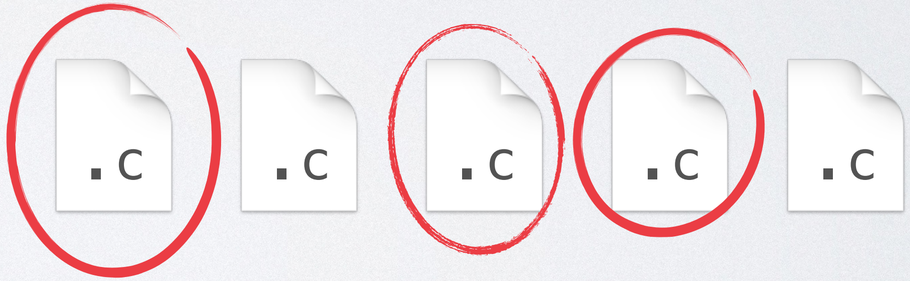
```
http_response
socket          +acc
...
```

compiler

```
lexeme
ast             +acc
...
```

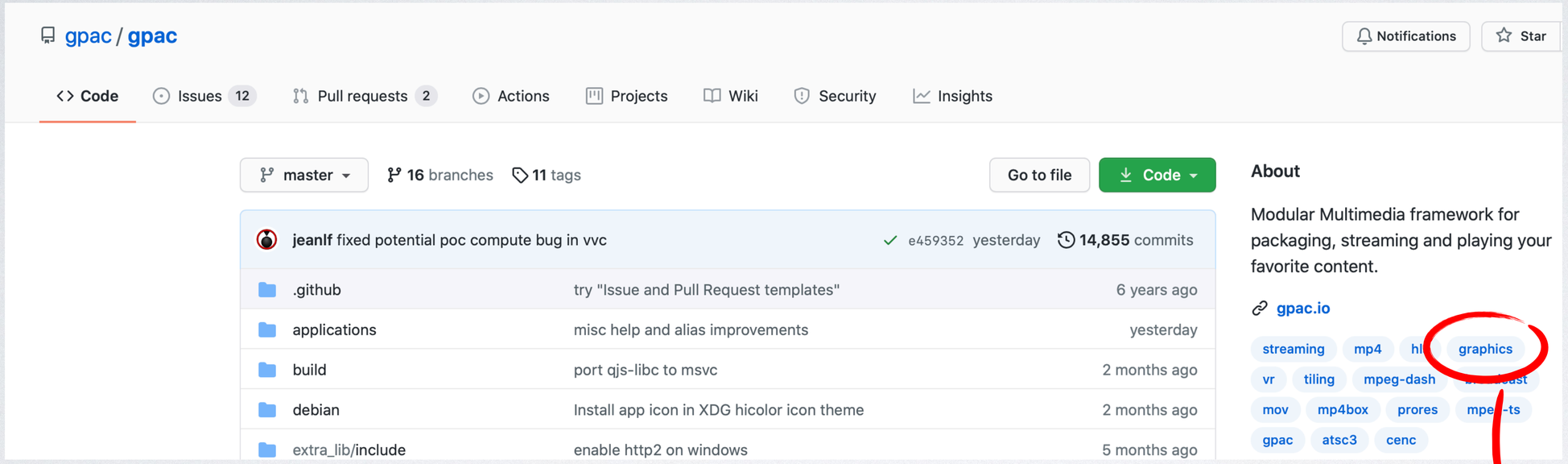
## Dataset Quality

Carefully selecting training data to improve generalizability.



Subproblem - Given a binary, how can we identify its domain?

# We derived domain labels from **GitHub** tags.

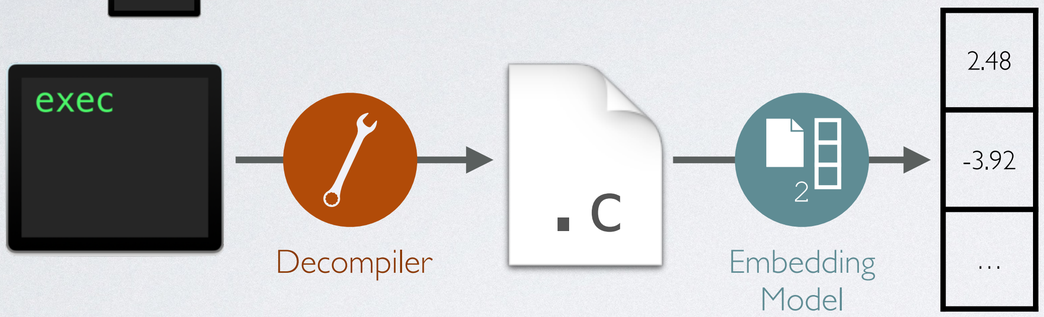



| Selected Domains |
| --- |
| kernel |
| game |
| shell |
| network |
| compiler |
| embedded |
| graphics |

Hand-selected 7 domains from among the most common GitHub tags in our dataset

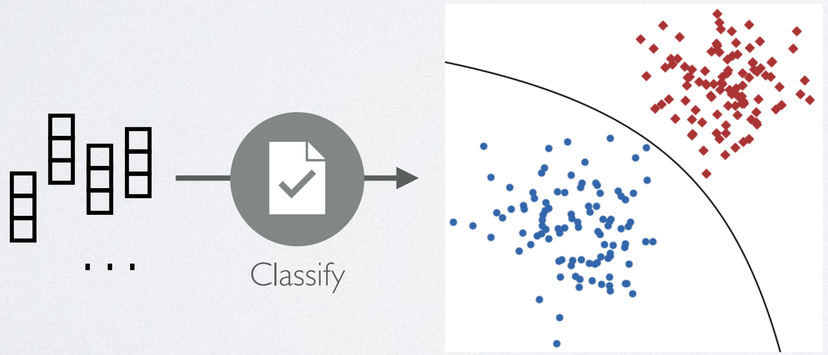
# We built a simple classifier to predict software domain from individual binaries.

For Each 

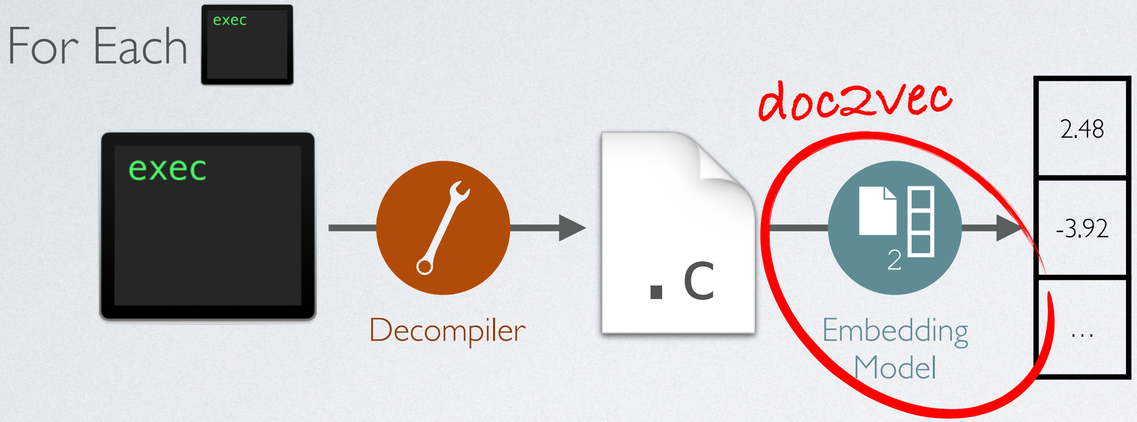




Strategy:

Then

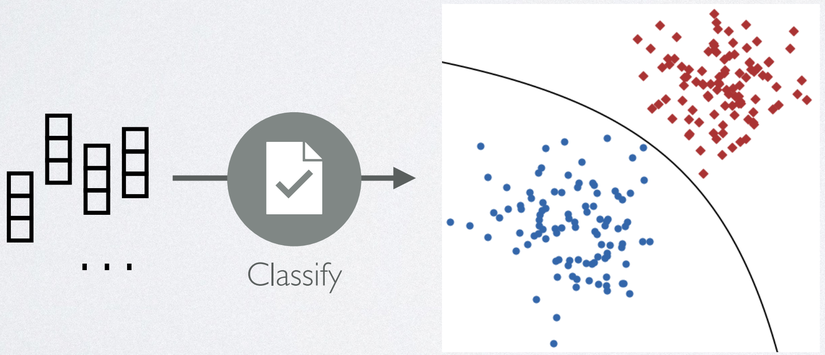


# We built a simple classifier to predict software domain from individual binaries.

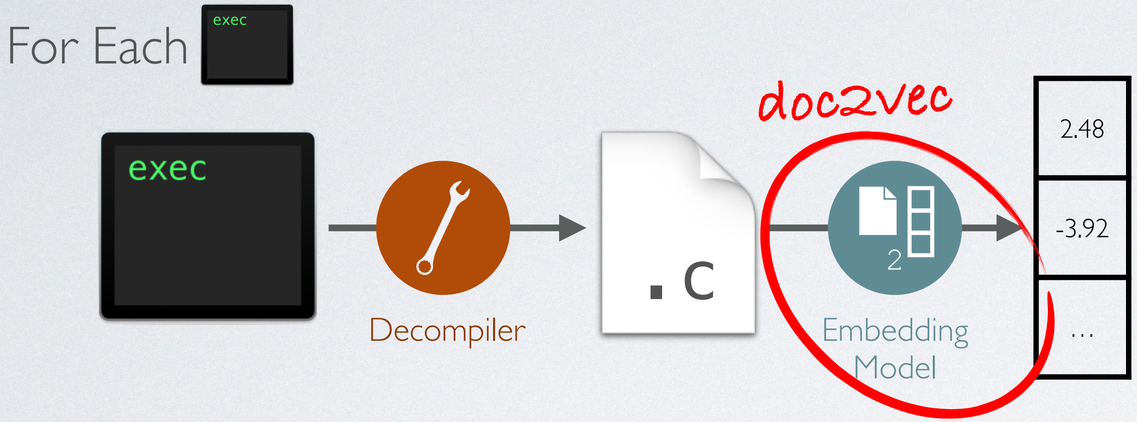



Strategy:

Then

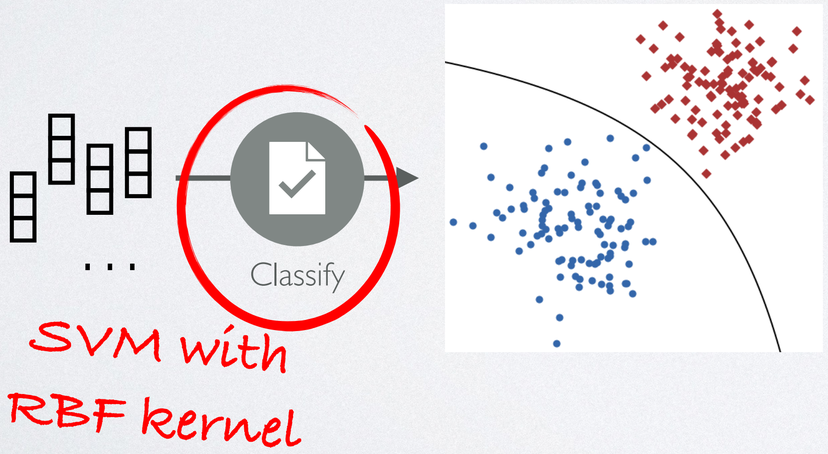


# We built a simple classifier to predict software domain from individual binaries.

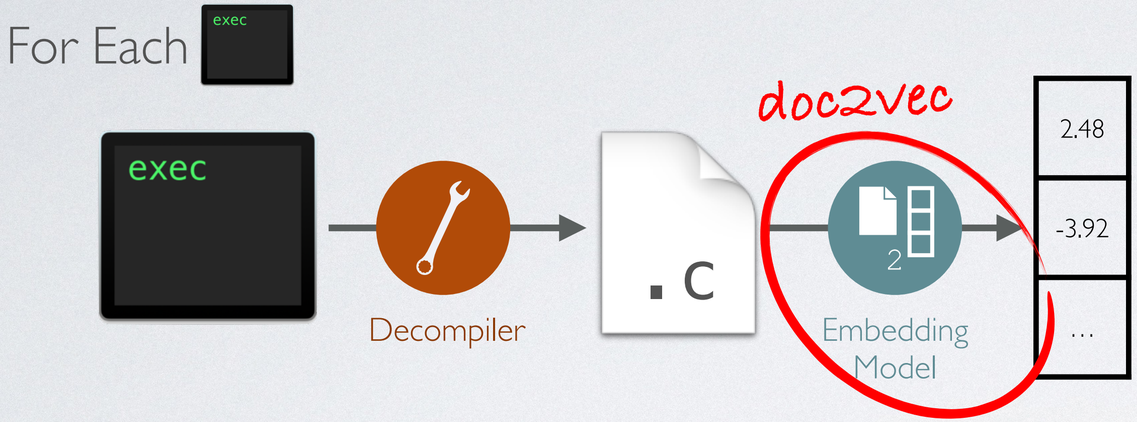



Strategy:

Then



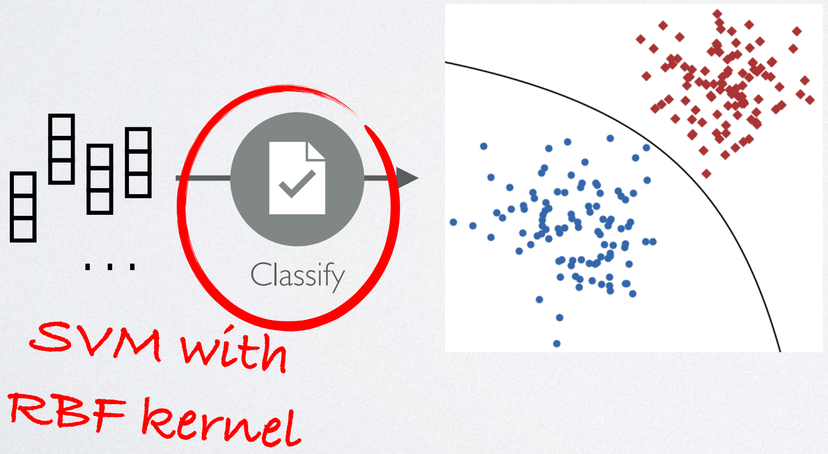
# We built a simple classifier to predict software domain from individual binaries.



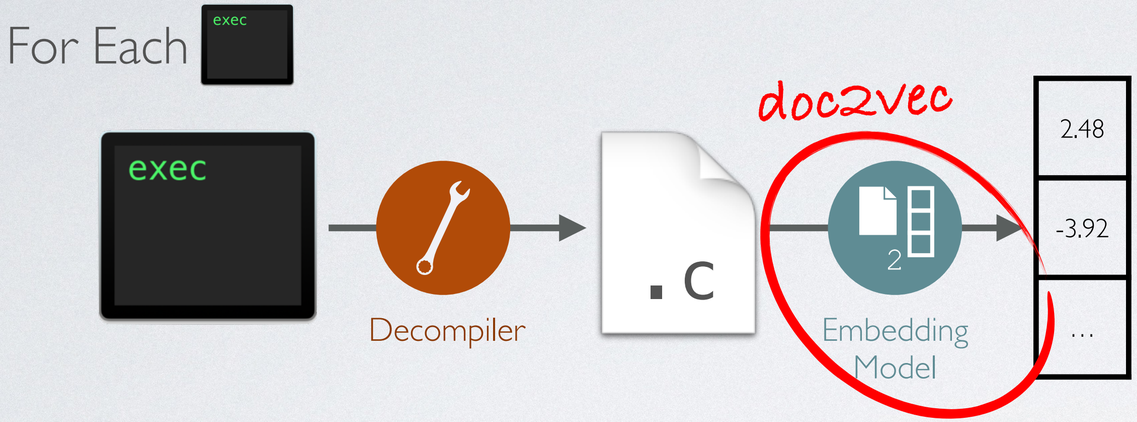

Strategy:

Remove parts of the code

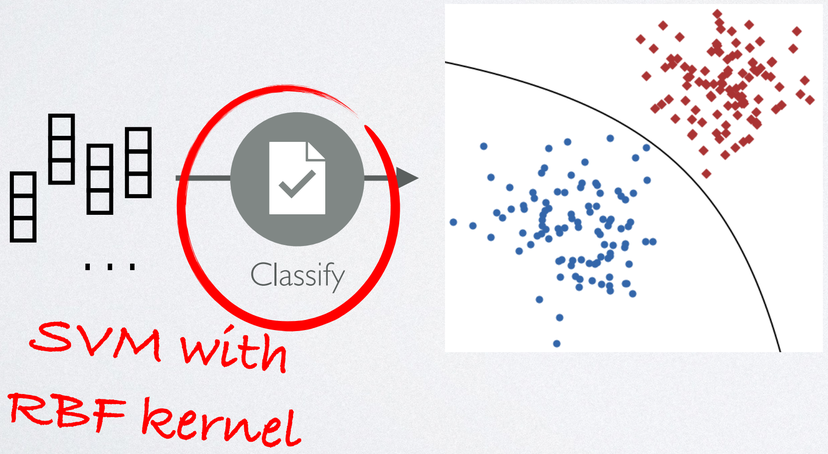
Then



# We built a simple classifier to predict software domain from individual binaries.



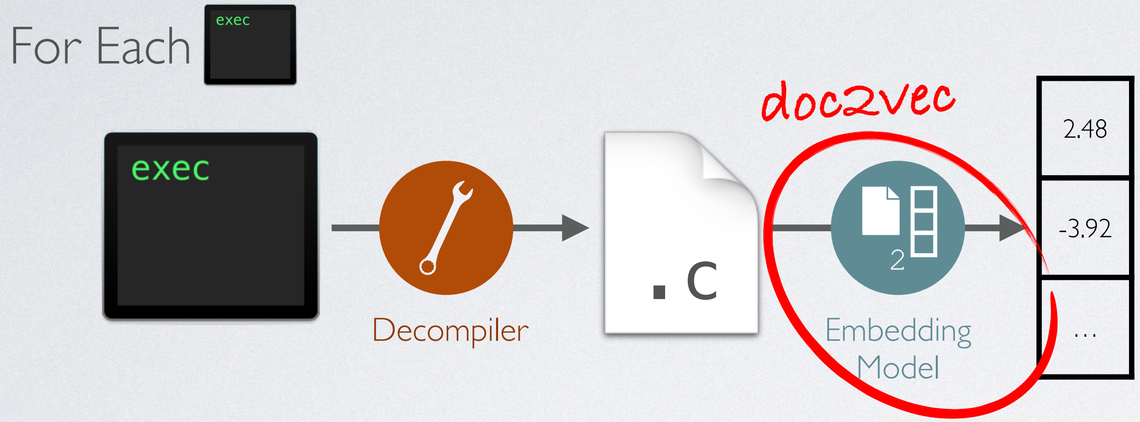

Then



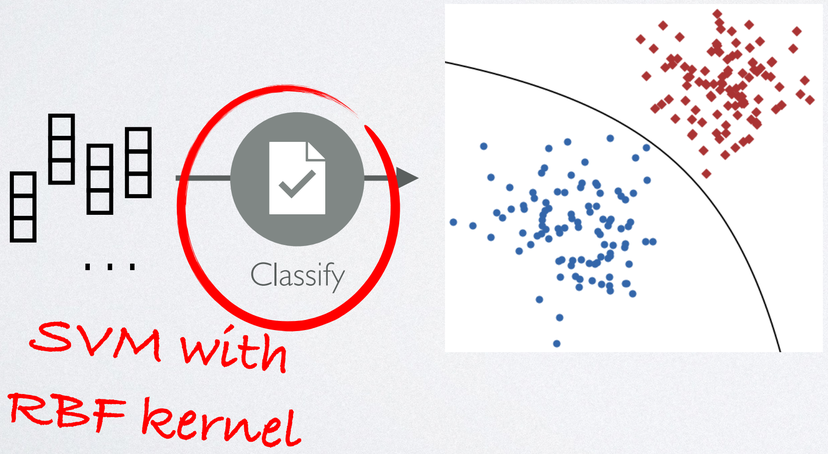

Strategy:

- Remove parts of the code
- Create Classifier

# We built a simple classifier to predict software domain from individual binaries.



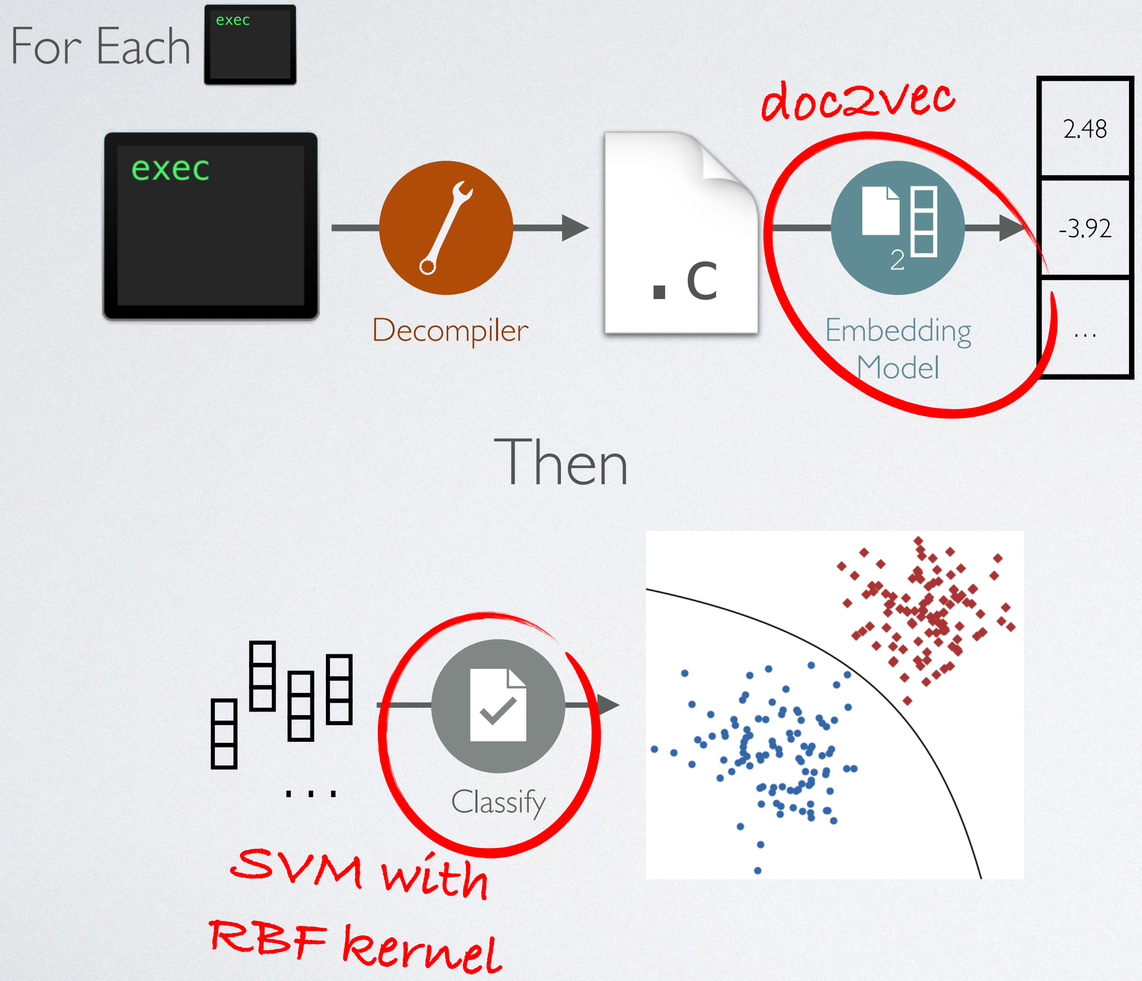

Then




Strategy:

- Remove parts of the code
- Create Classifier
- Evaluate

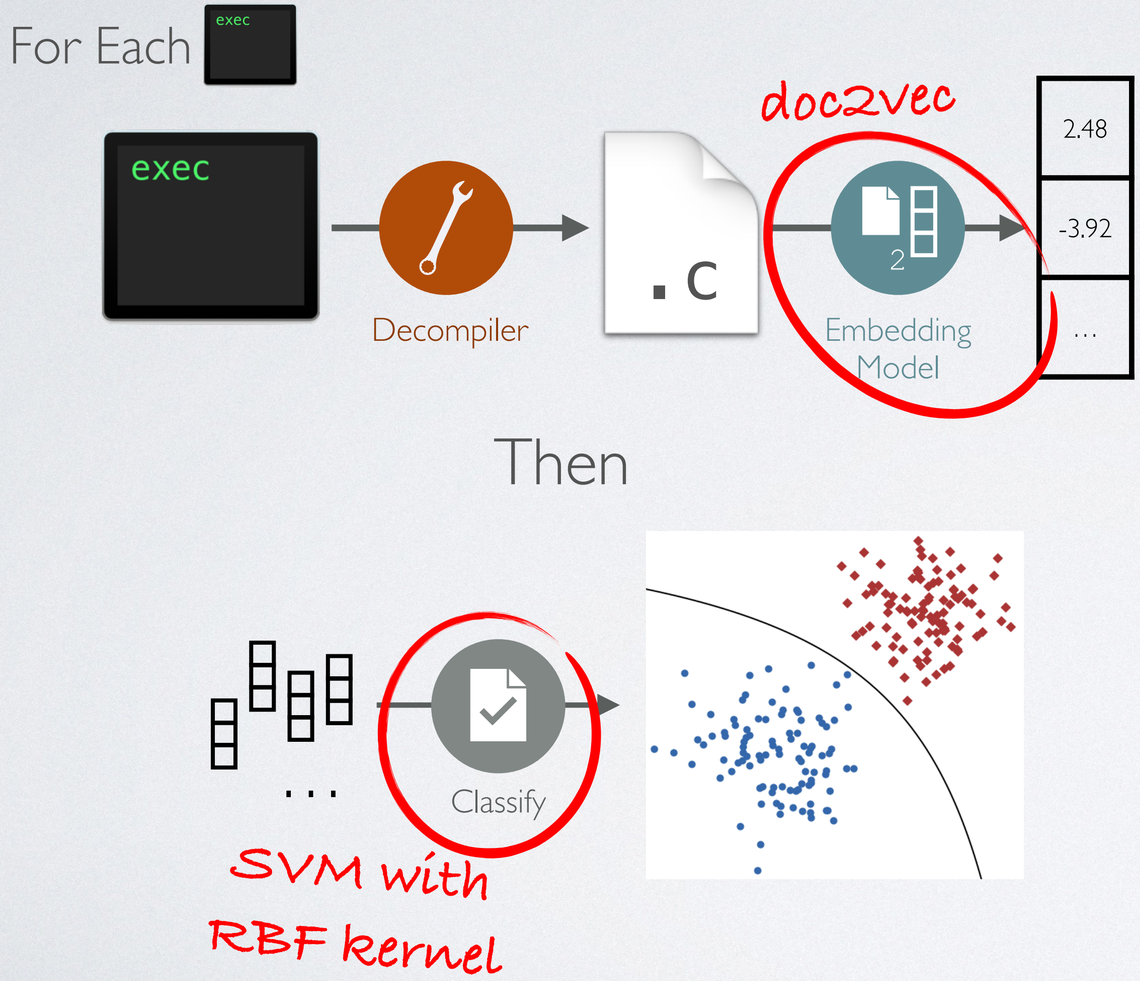
# We built a simple classifier to predict software domain from individual binaries.




Strategy:

- Remove parts of the code
- Create Classifier
- Evaluate
- Repeat

# We built a simple classifier to predict software domain from individual binaries.



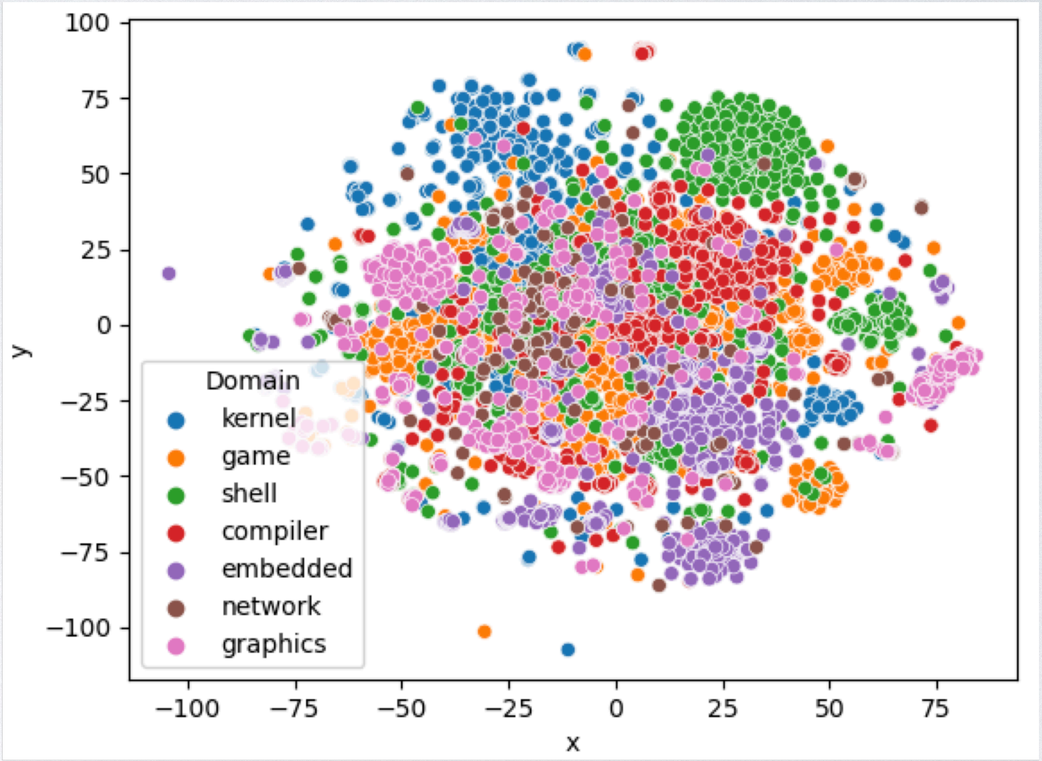

Strategy:

- Remove parts of the code
- Create Classifier
- Evaluate
- Repeat

Observe which parts of code are most predictive of software domain.

# In aggregate, string literals are highly indicative of software domain.

- The approach that led to the best performance was to extract the string literals run the classifier only on the string literals.
- Test Accuracy: 83.54%




Dimensionality-reduced visualization of embeddings generated (originally 64 dimensions)

We now know that string literals inform domain.

We can use this information to make DIRE predict domain-specific identifiers more accurately

# We modified the DIRE model to use string literals directly instead of placeholders.

**Orignal DIRE Model**

```
result = gf_log(“String");
```

**Domain-Aware DIRE Model**

```
result = gf_log("[Compositor] Failed to allocate svg gradient stack\n");
```

# We borrowed categorical labeling techniques from machine translation to help inform our model of the domain of the data.

| **Machine Translation** | **DIRE's Variable Renamings** |
| --- | --- |
| <english> I went to the store. | <graphics> __int64 __fastcall compositor_init_s… |
| <japanese> 私は店に行った。 | <compiler> unsigned __int64 *__fastcall endian… |

Labeled data is rare ⟶ Train mostly on data with no known domain (label `<unknown>`)

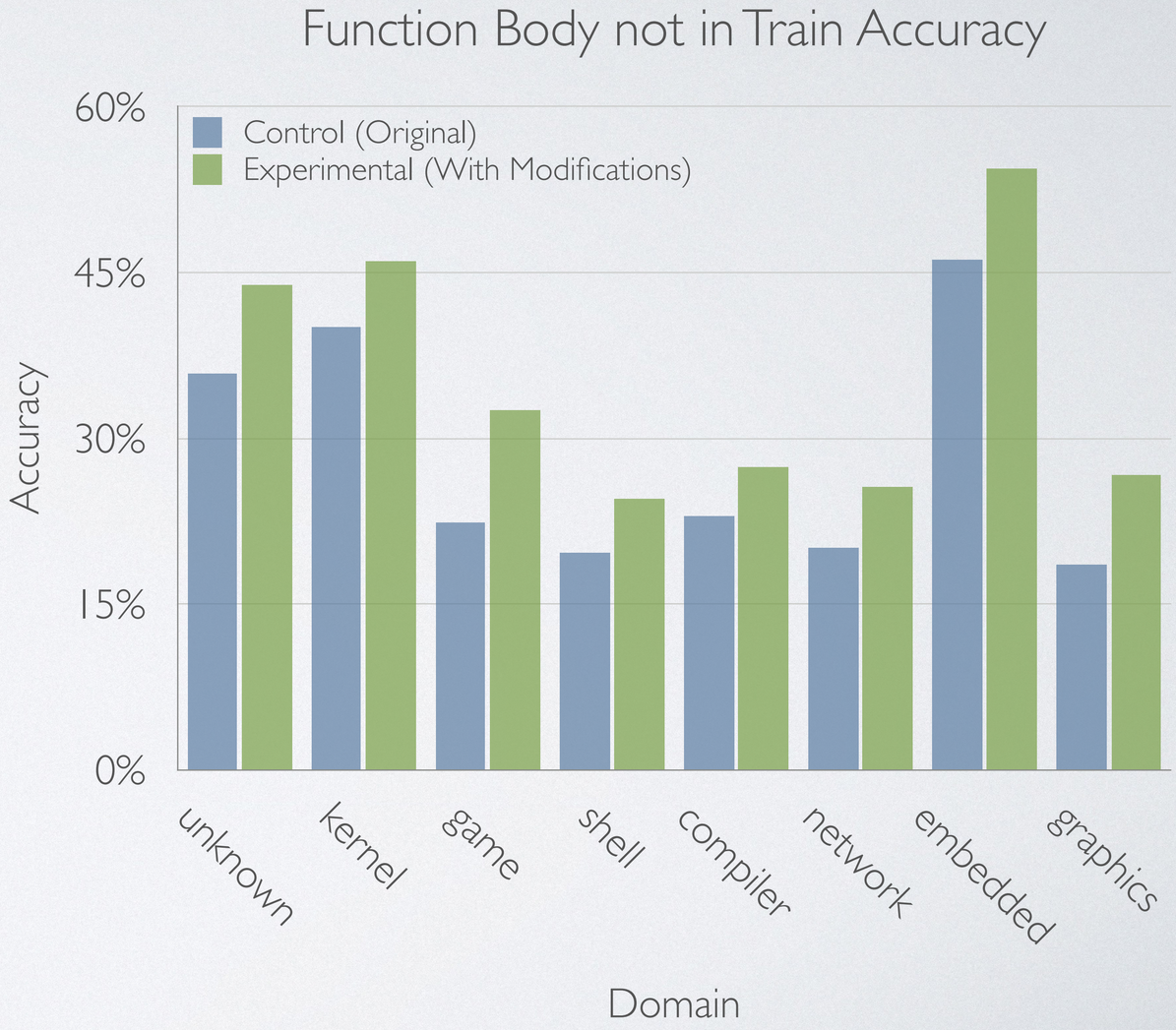
# Each modification applied to DIRE's data serves a different purpose.

| | |
|---|---|
| Domain labels | Provides general, *global* information about software domain to each function in the binary (even those without string literals). |
| String Literals | Provides more *specific* information about a particular function's role and/or the role of certain variables in that function. |

# With these modifications, DIRE performs better across all domains.

Function Body not in Train Accuracy

| Domain | Control (Original) | With Modifications | Accuracy Increase |
|---|---|---|---|
| **unknown** | 35.89% | 43.85% | 7.96% |
| **kernel** | 39.99% | 46.07% | 6.08% |
| **game** | 22.40% | 32.51% | 10.11% |
| **shell** | 19.63% | 24.49% | 4.86% |
| **compiler** | 22.97% | 27.42% | 4.45% |
| **network** | 20.13% | 25.60% | 5.47% |
| **embedded** | 46.19% | 54.33% | 8.14% |
| **graphics** | 18.58% | 26.67% | 8.09% |
| **Mean (Domains)** | **27.13%** | **33.87%** | **6.74%** |



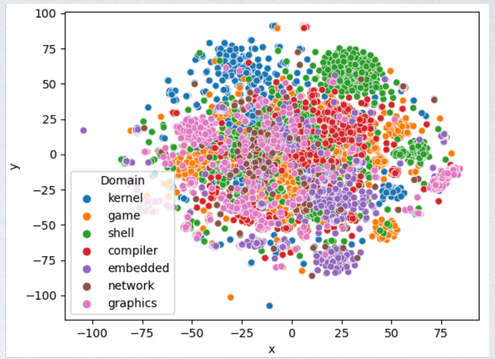
# We improved DIRE to predict domain-specific variable names more accurately.

ML models can struggle on rare or underrepresented concepts like domain specific variable names.

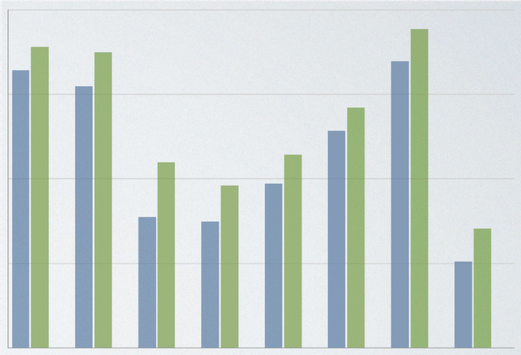
<graphics> __int64 __fastcall compositor_init_s…

<compiler> unsigned __int64 *__fastcall endian…

We modified DIRE's data to help it perform better on domain-specific variable names.




In aggregate, string literals are highly indicative of a given binary's software domain.

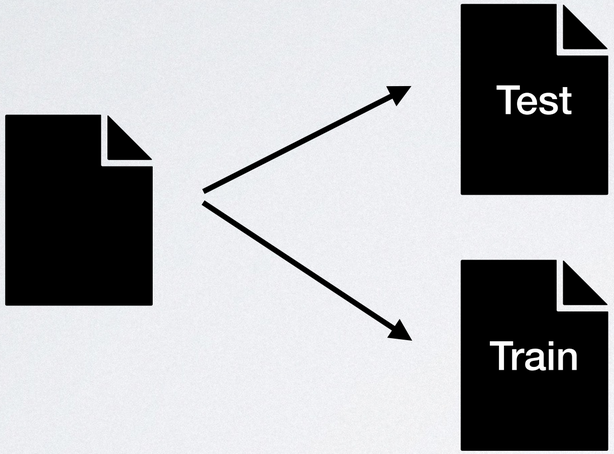


The helps DIRE consistently outperform the original version.

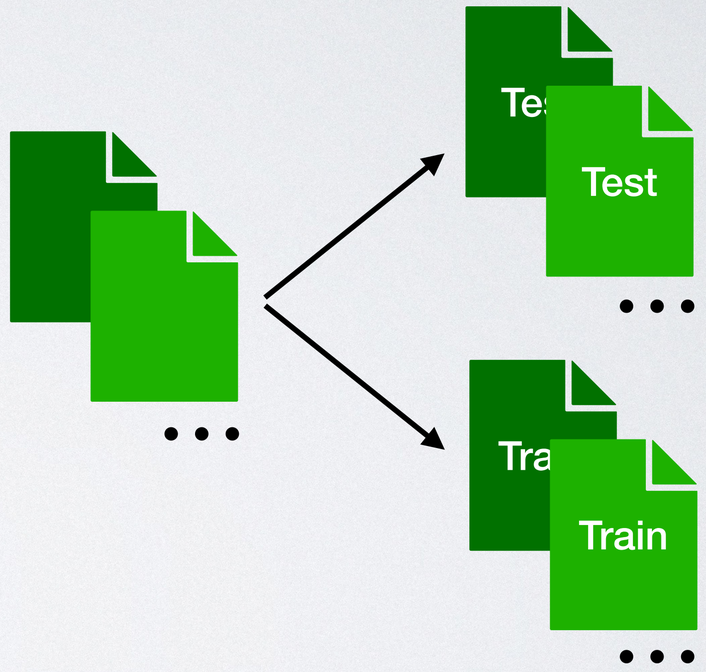
# Bonus Slides - Methodology

# We split each dataset into train and test sets.

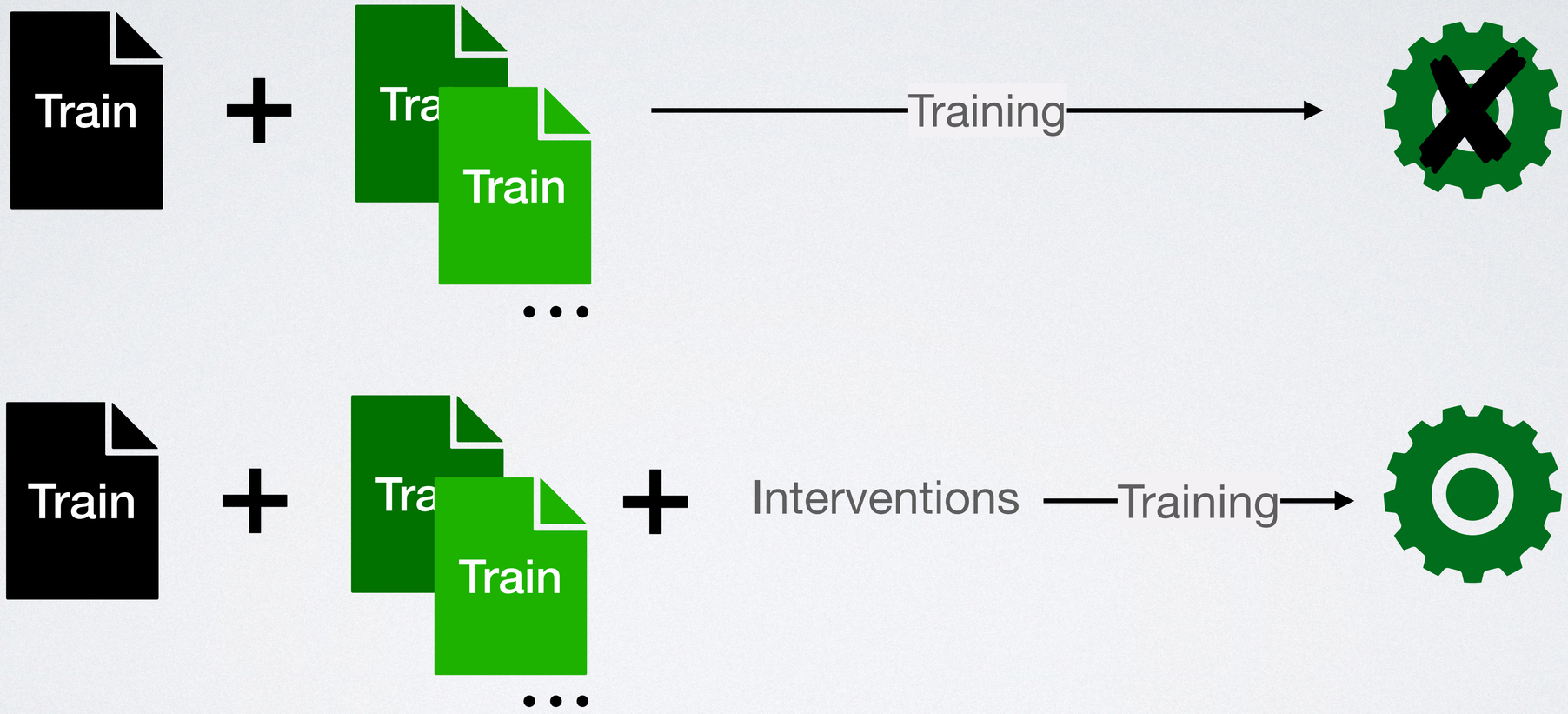
Unknown data



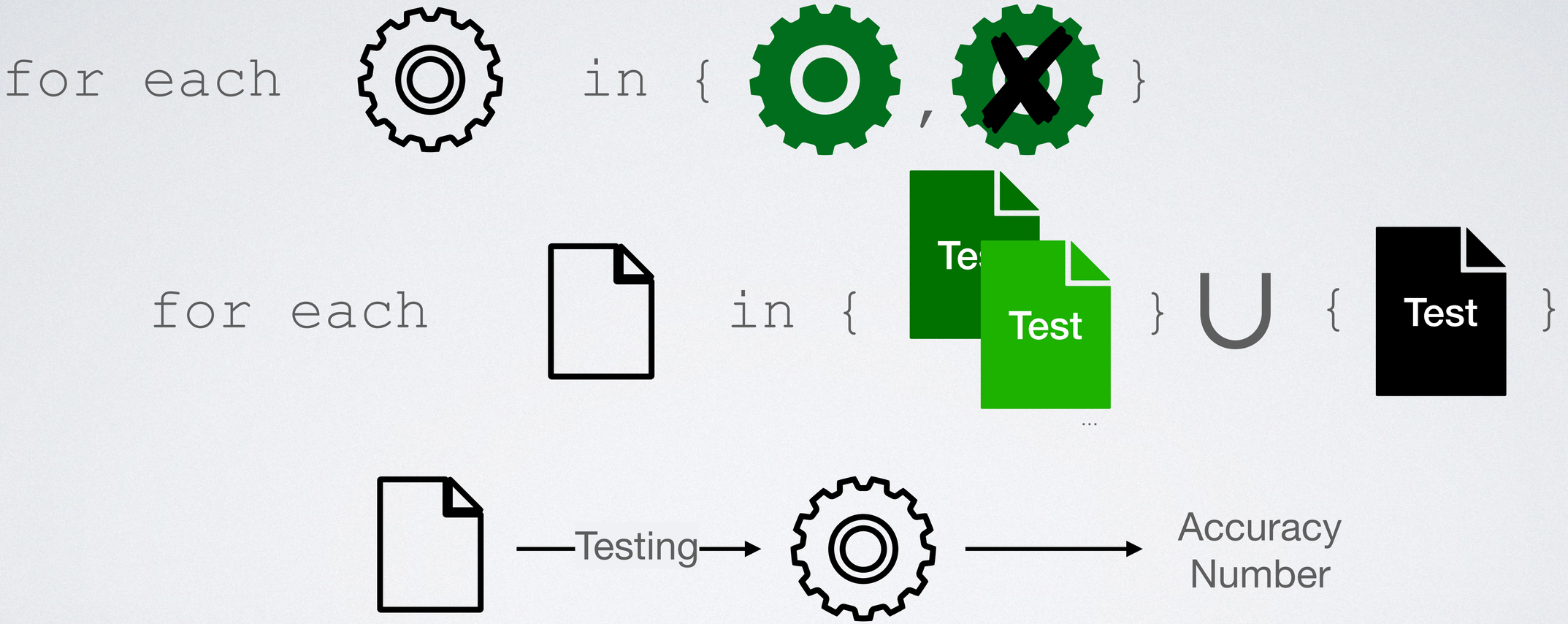

Domain-specific Data (kernel, network, etc.)



# We train a control model and a model with interventions applied.



# We test each model against each test set.



# Bonus Slides - Training Set Quality

How do we select "good" training data for DIRE and machine learning models of code in general?

# Variable names are *labels* for concepts present in code

```
listNode *
listSearchKey(list *list, void *key) {
    listIter iter;
    listNode *node;

    listRewind(list, &iter);
    while((node = listNext(&iter)) != NULL) {
        if (list->match) {
            if (list->match(node->value, key)) {
                return node;
            }
        } else {
            if (key == node->value) {
                return node;
            }
        }
    }
    return NULL;
}
```

Entities: **`list`**, **`key`**, **`iter`**, etc.

```
ngx_buf_t *
ngx_create_temp_buf(ngx_pool_t *pool, size_t size) {
    ngx_buf_t *b;

    b = ngx_calloc_buf(pool);
    if (b == NULL) {
        return NULL;
    }

    b->start = ngx_palloc(pool, size);
    if (b->start == NULL) {
        return NULL;
    }

    b->pos = b->start;
    b->last = b->start;
    b->end = b->last + size;
    b->temporary = 1;

    return b;
}
```

Entities: **`pool`**, **`size`**, **`start`**, etc.

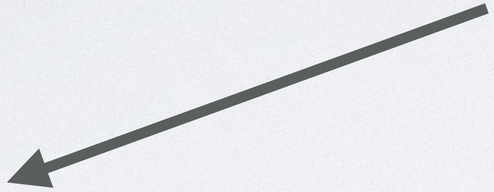
***Hypothesis***: Training data with a wider variety of variable names in code will lead to a more generalizable model.

***Hypothesis***: Training data with a wider variety of variable names in code will lead to a more generalizable model.

***Names*** are ***labels*** for the entities present in code

***Hypothesis***: Training data with a wider variety of variable names in code will lead to a more generalizable model.

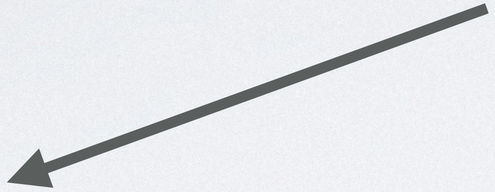
***Names*** are ***labels*** for the entities present in code



More ***diverse*** labels

***Hypothesis***: Training data with a wider variety of variable names in code will lead to a more generalizable model.

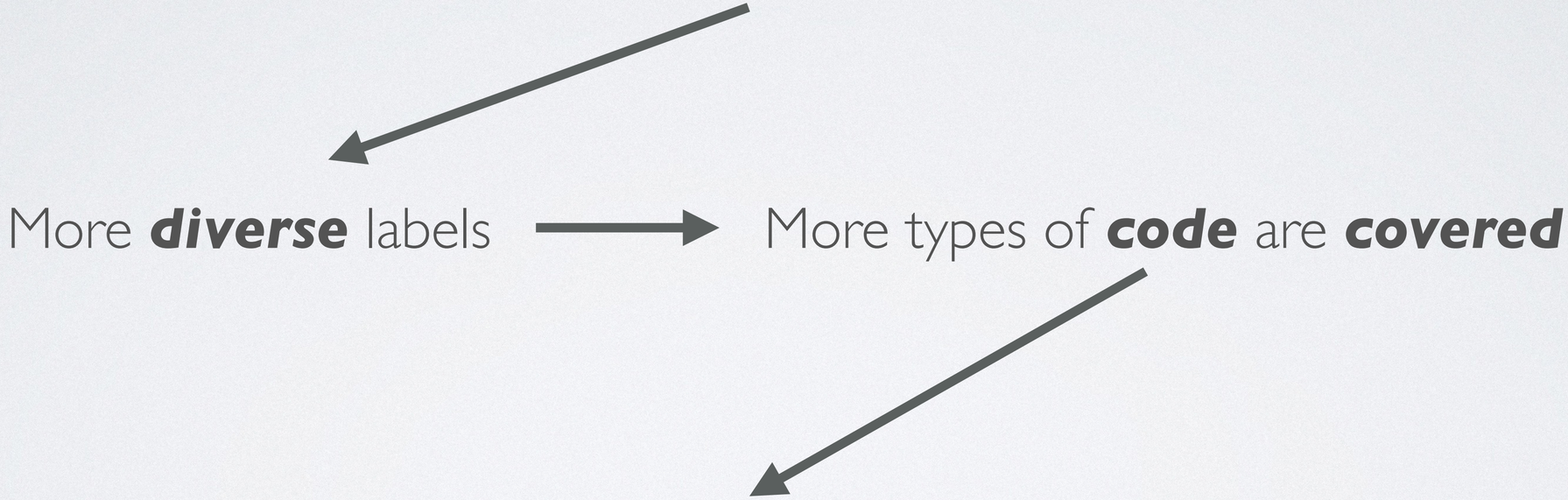
***Names*** are ***labels*** for the entities present in code



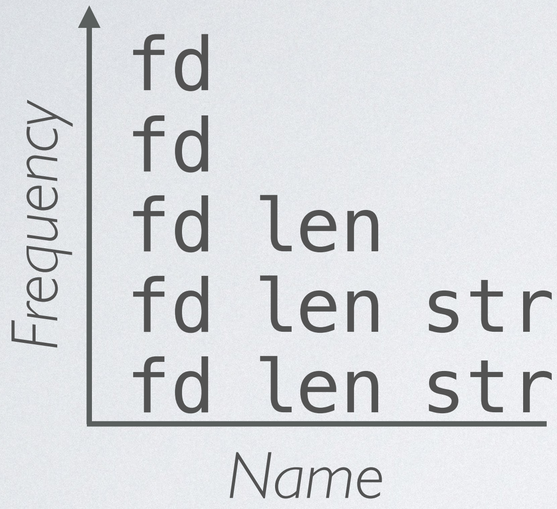
More ***diverse*** labels ⟶ More types of ***code*** are ***covered***

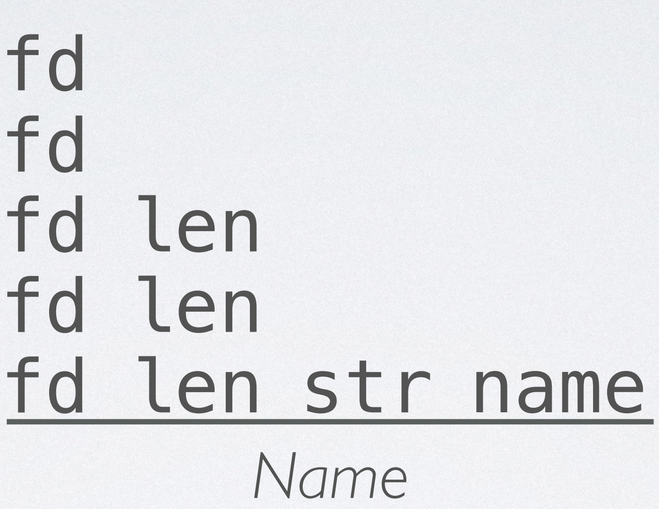
***Hypothesis***: Training data with a wider variety of variable names in code will lead to a more generalizable model.

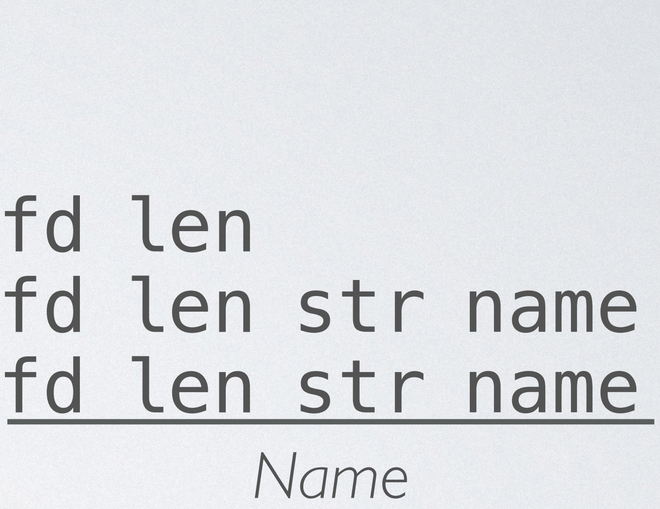
***Names*** are ***labels*** for the entities present in code



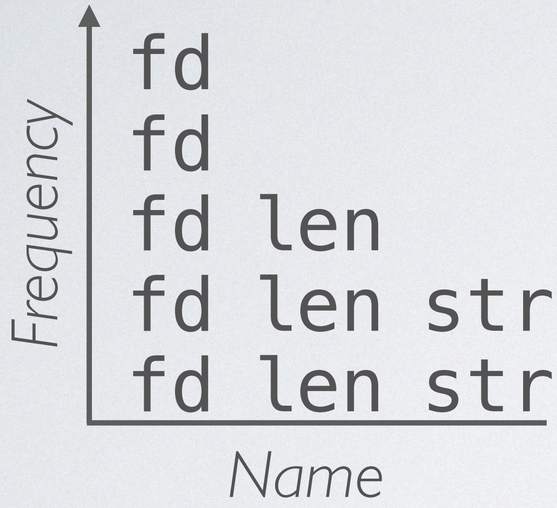
# We measure training set diversity with entropy.

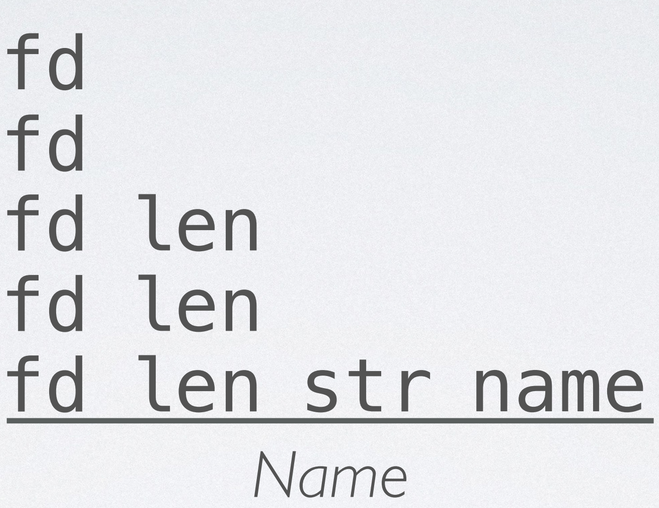


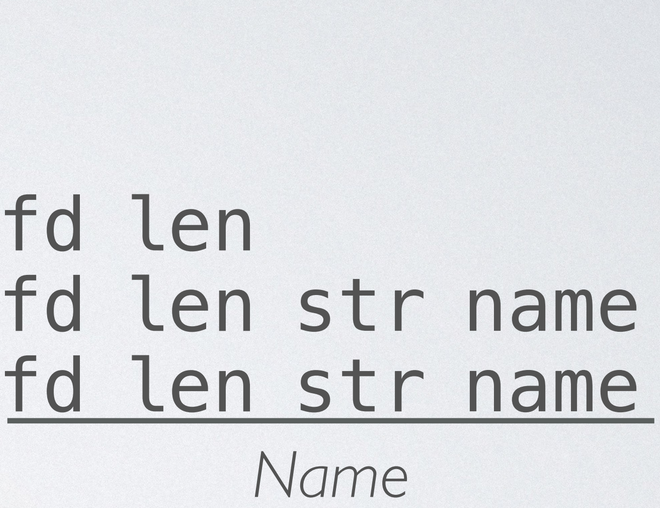



# We measure training set diversity with entropy.



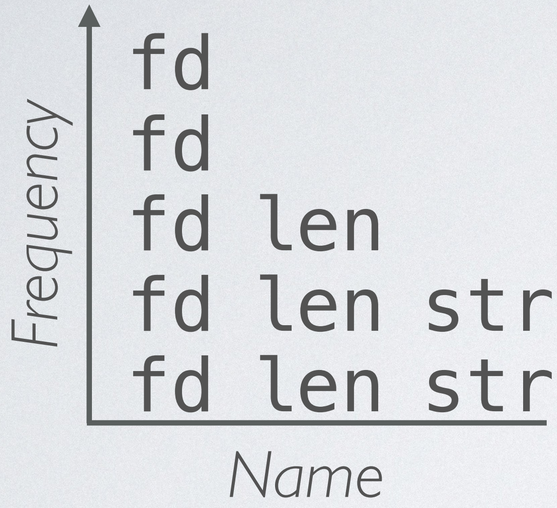




Entropy

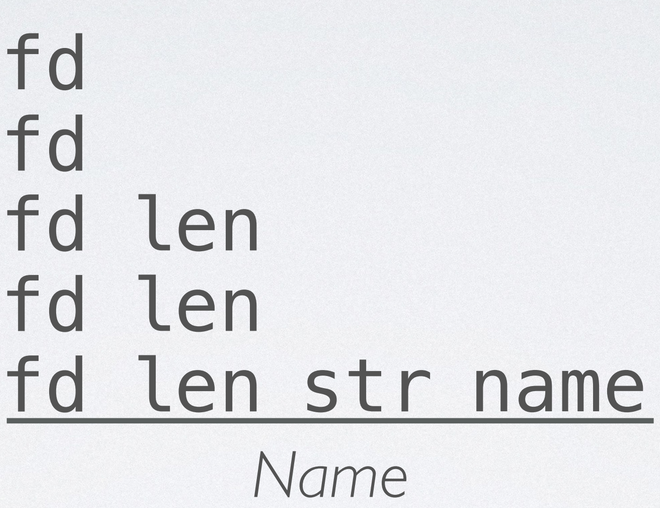
1.49 bits

# We measure training set diversity with entropy.



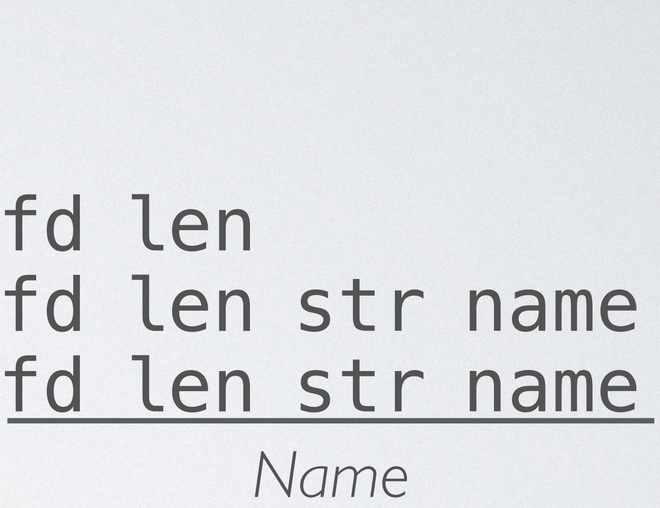

Entropy

1.49 bits

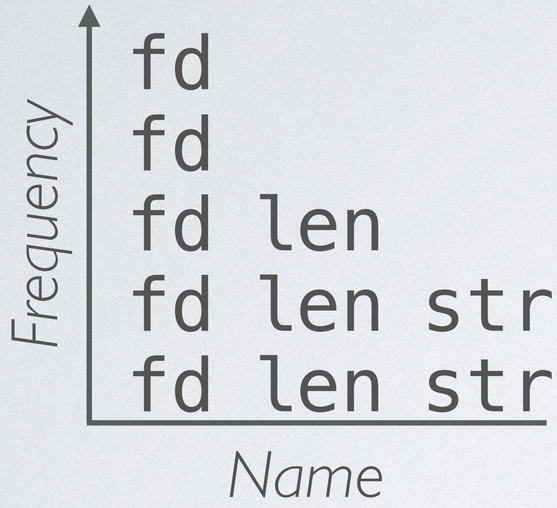



Entropy

1.69 bits

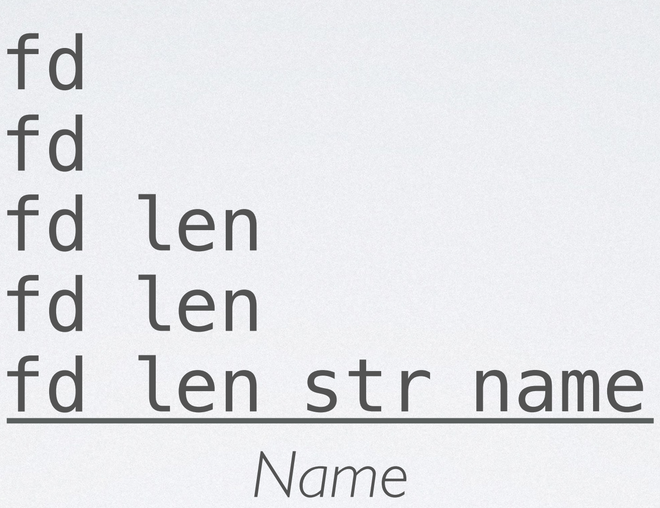


# We measure training set diversity with entropy.



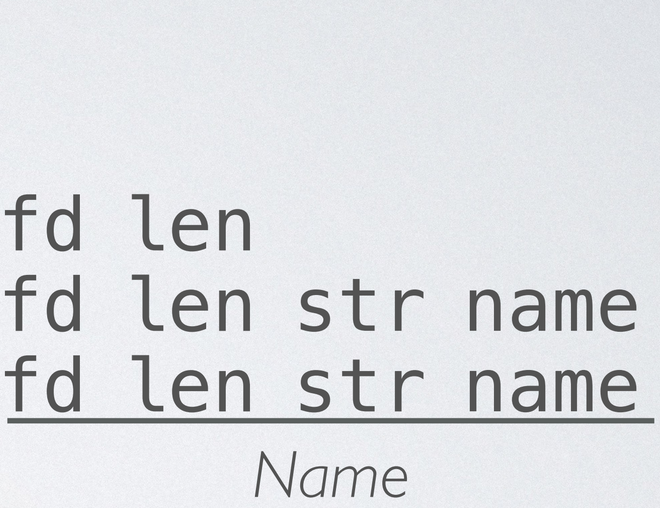

Entropy

1.49 bits




Entropy

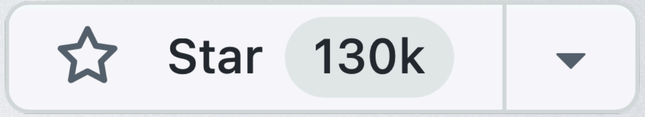
1.69 bits




Entropy

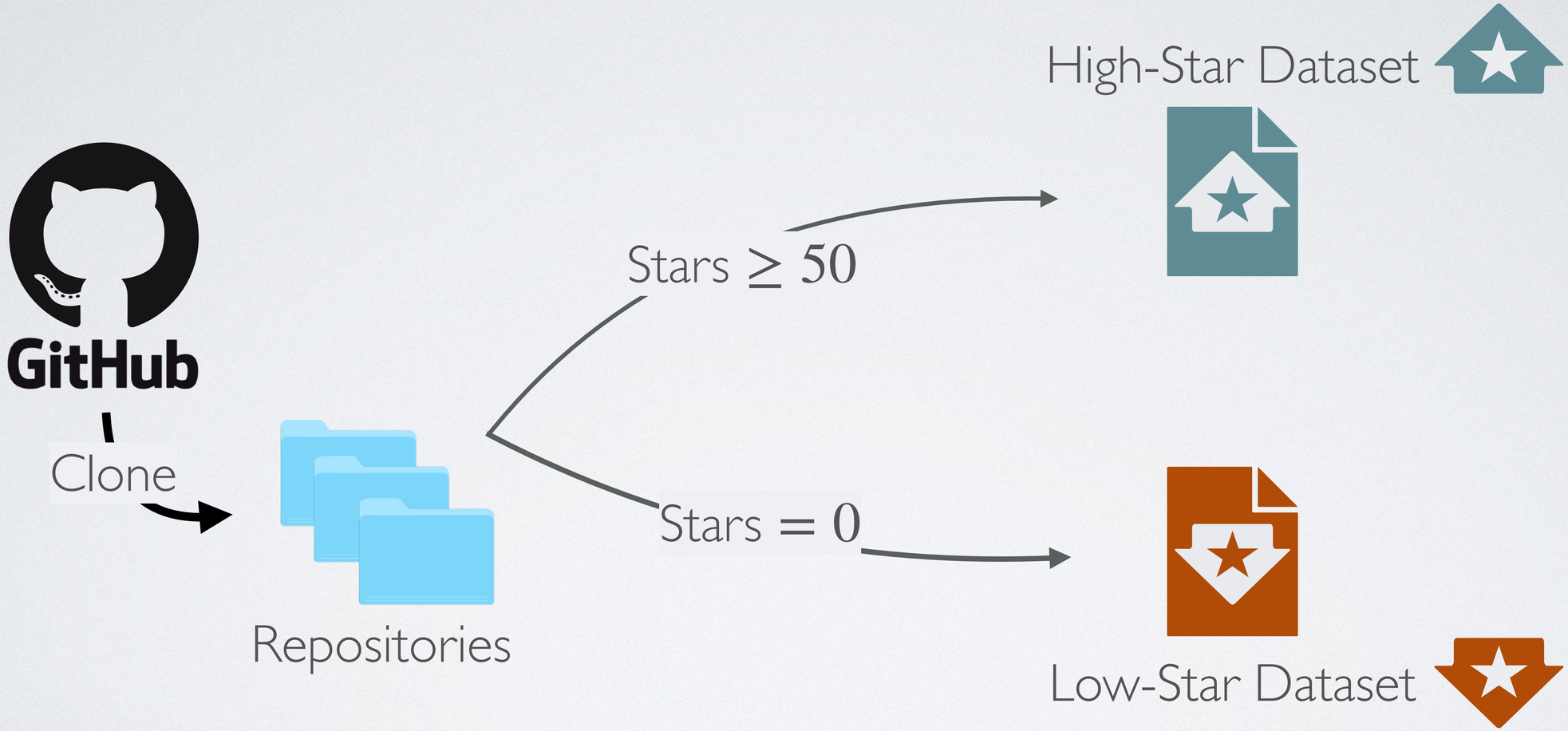
1.97 bits

To test our hypothesis, we tried training a model on code that has a high number of stars on GitHub

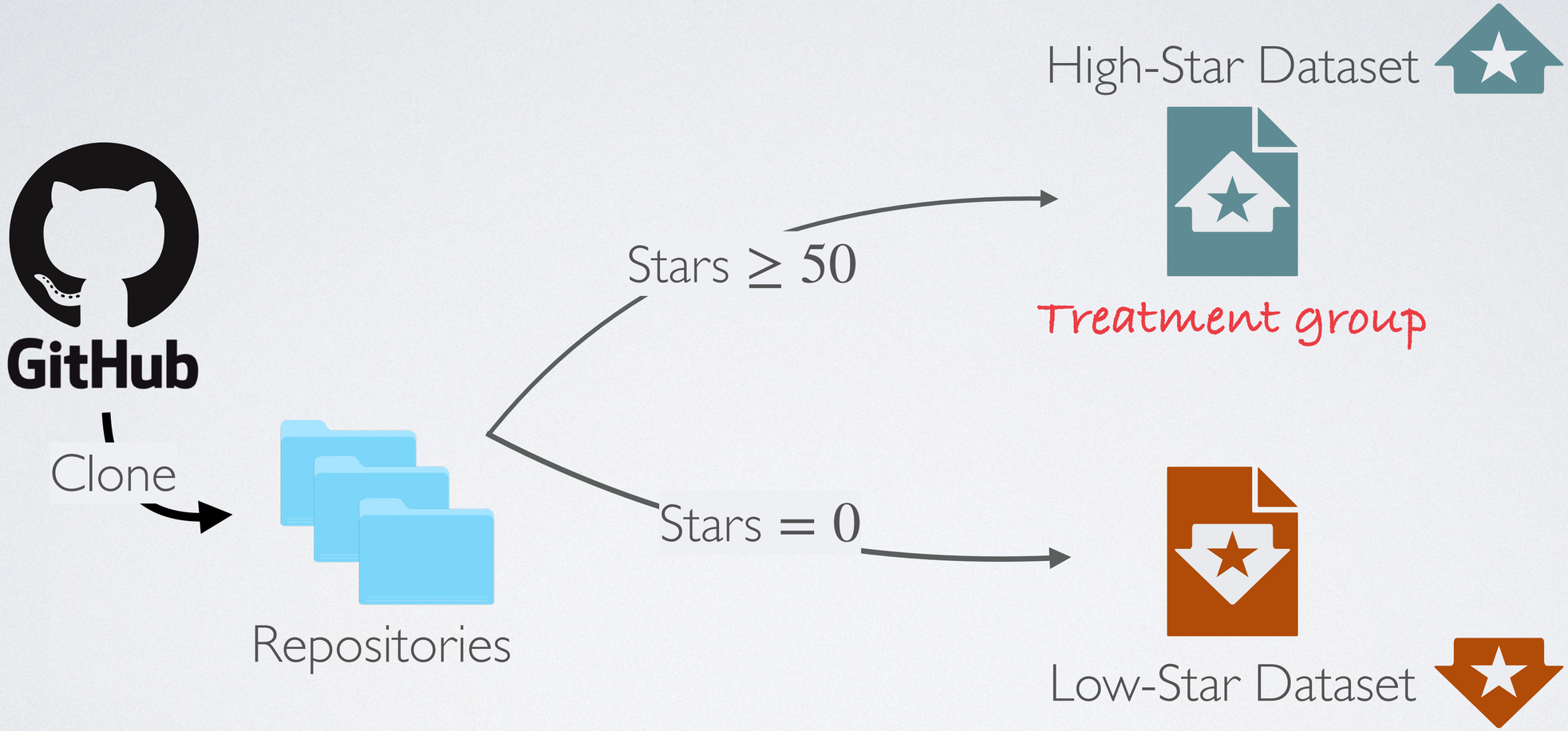



- We suspected high-star repositories likely have better names
- We suspected high-star repositories may be more diverse
- High-star repositories are much less likely to be duplicates

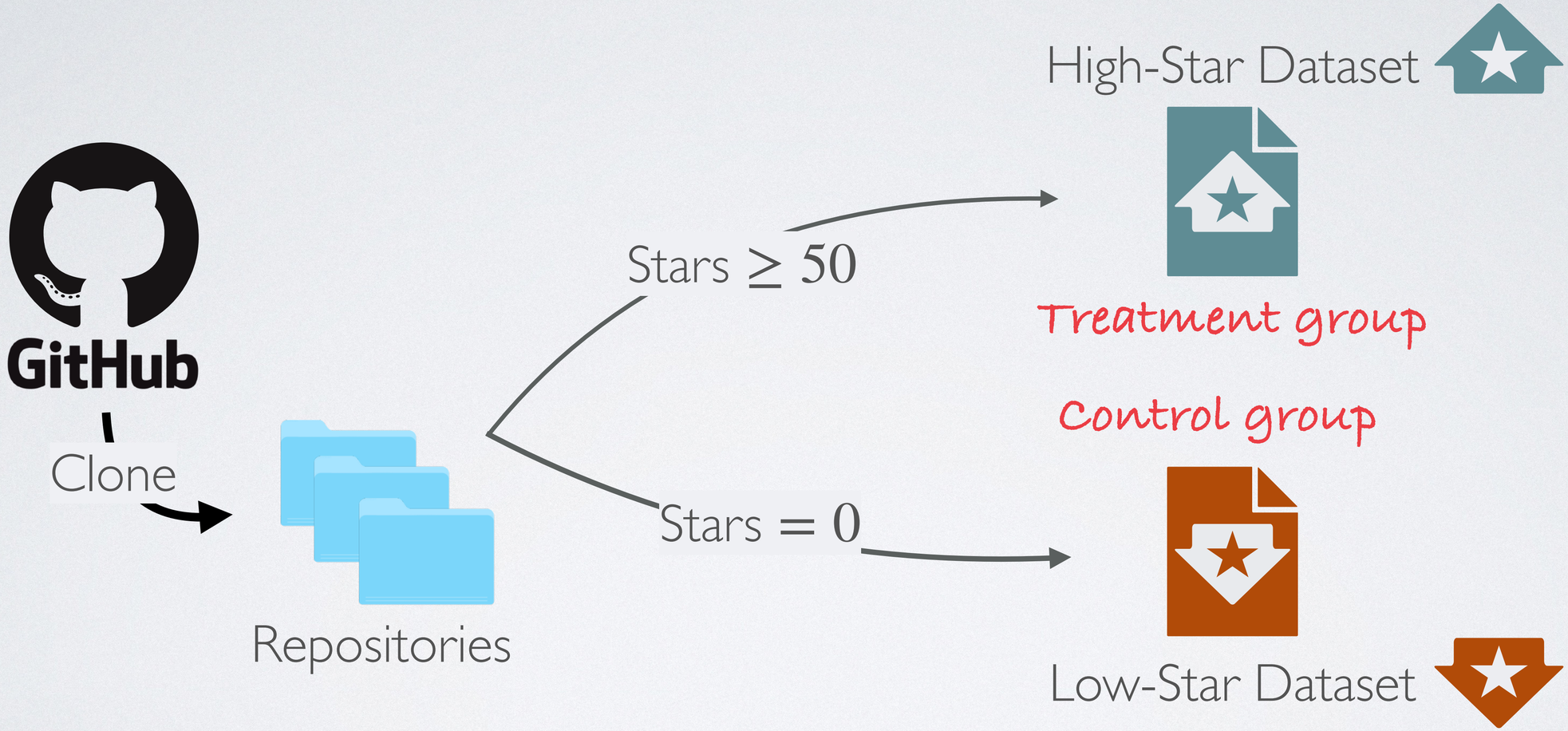
# We cloned repositories from **GitHub** and sorted them by star count.



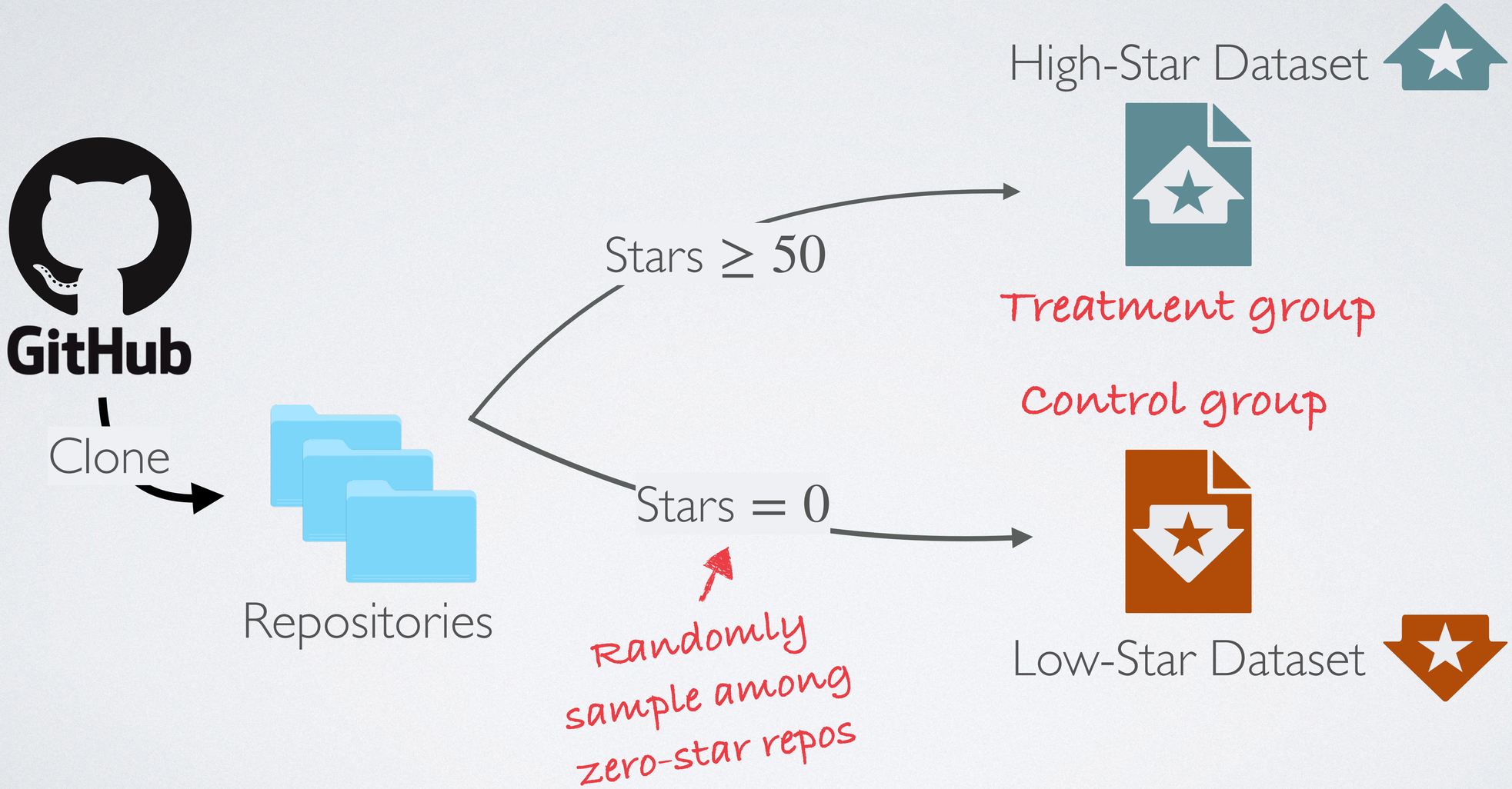
# We cloned repositories from **GitHub** and sorted them by star count.



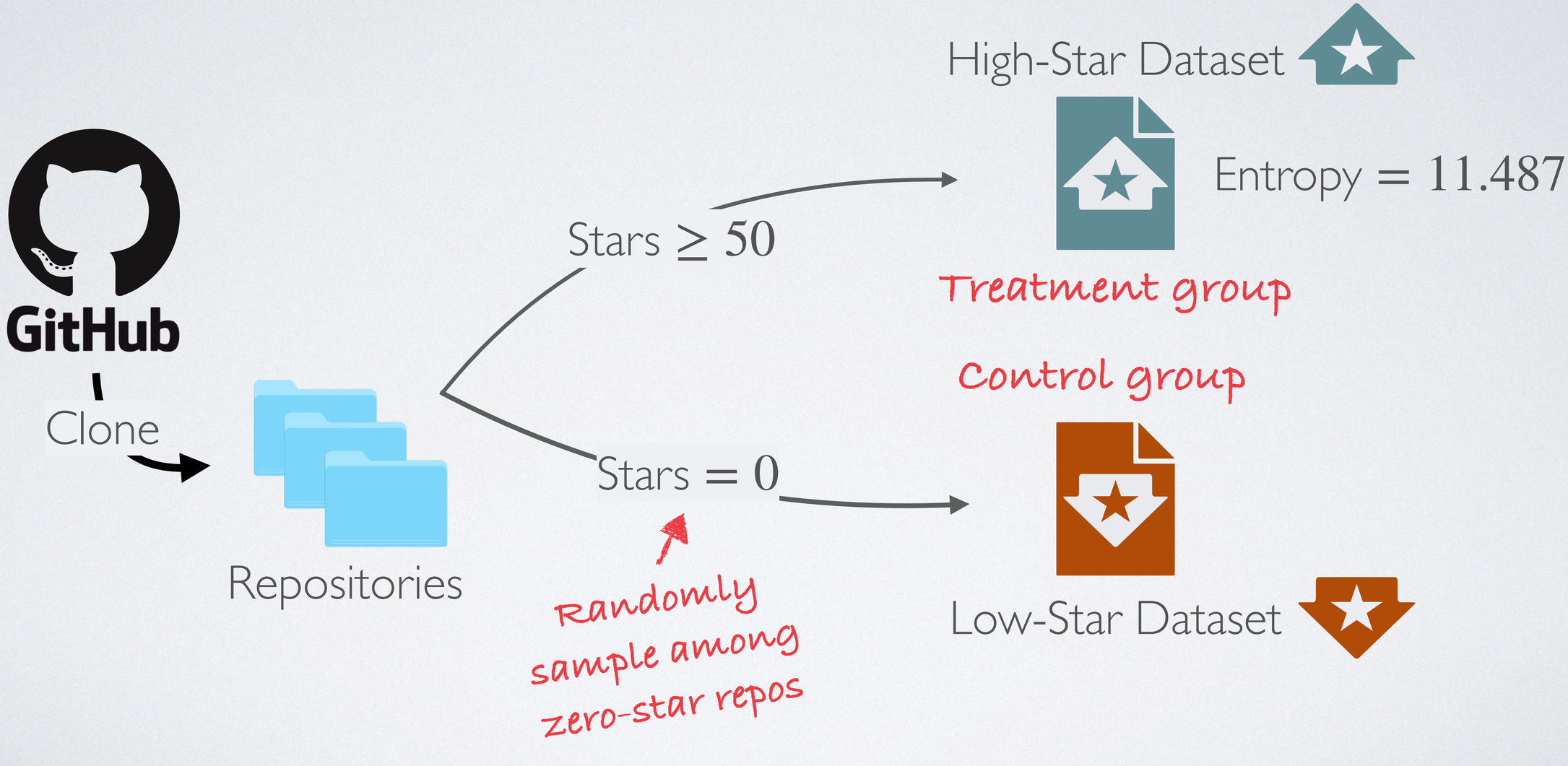
# We cloned repositories from **GitHub** and sorted them by star count.



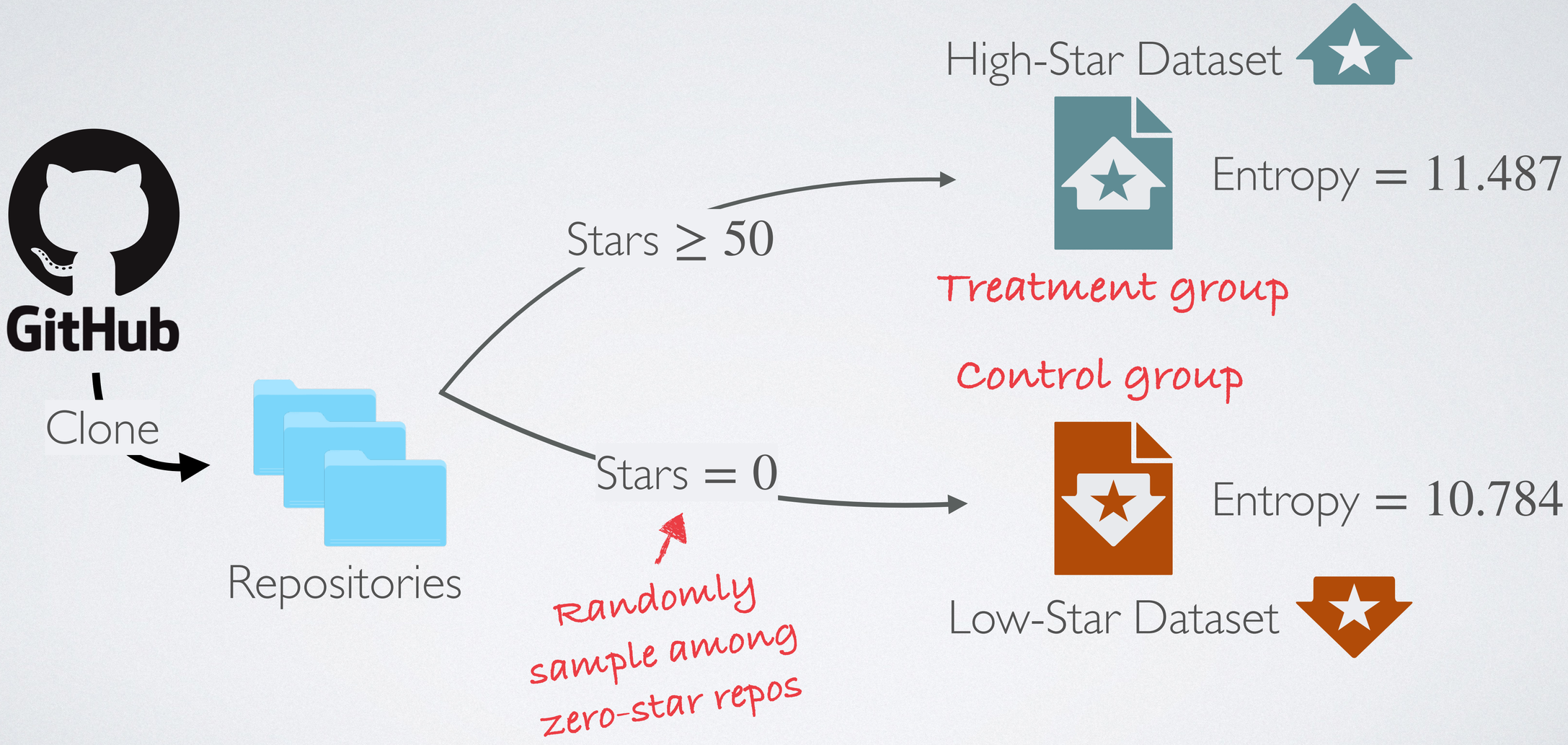
# We cloned repositories from **GitHub** and sorted them by star count.



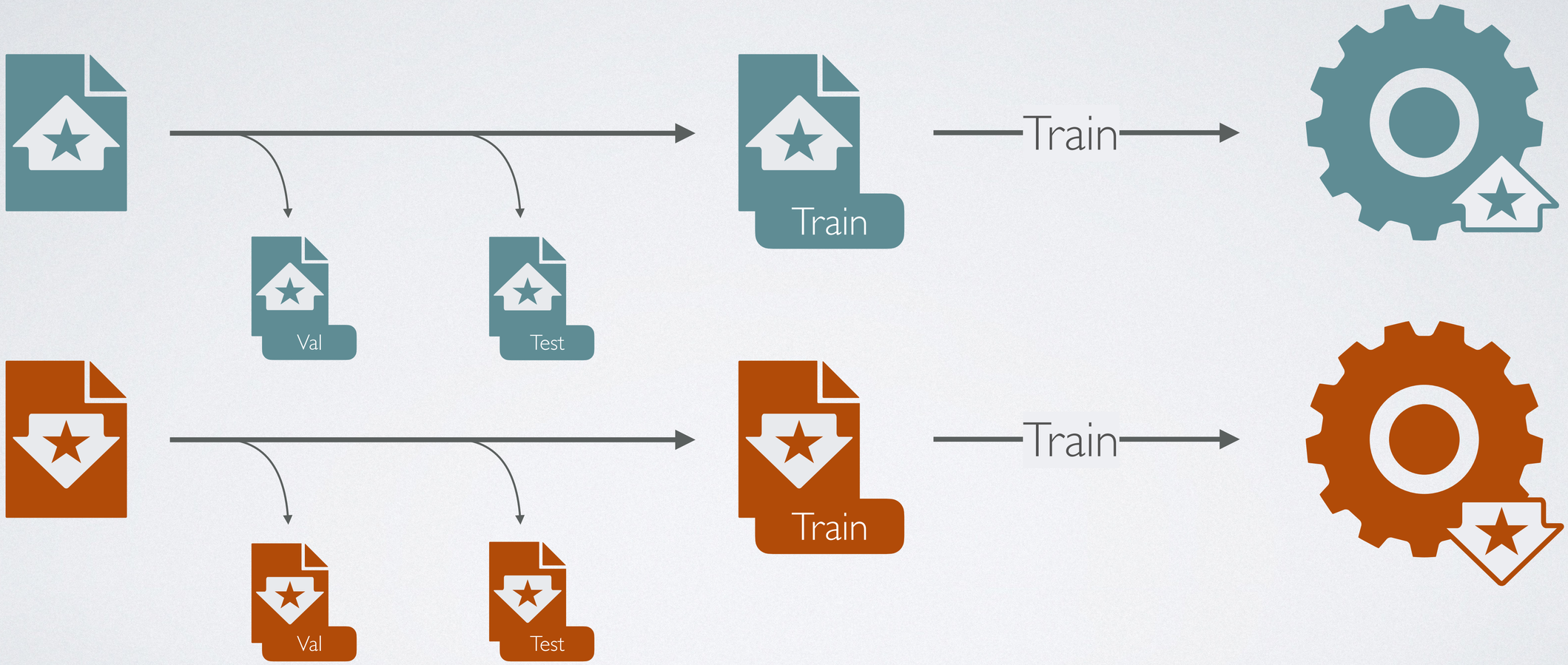
# We cloned repositories from **GitHub** and sorted them by star count.



# We cloned repositories from **GitHub** and sorted them by star count.

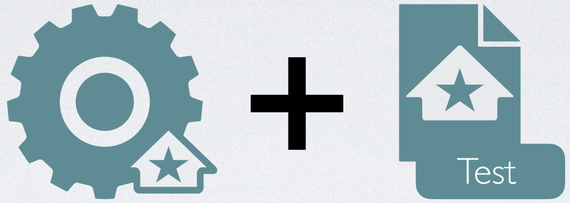


# After separating out testing and validation sets, we trained a DIRE model on high-star code and low-star code

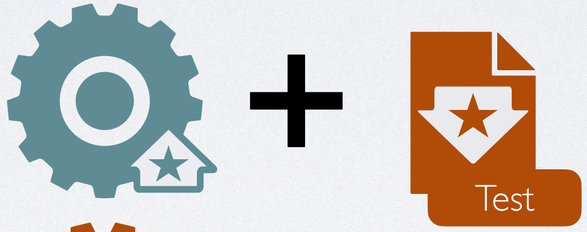


# We tested each model against each test set.

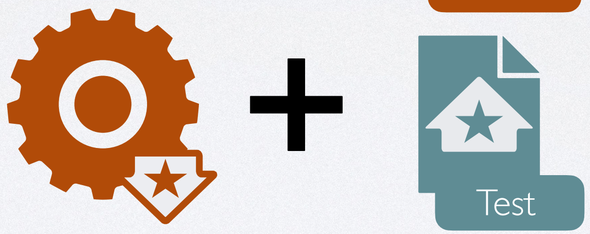
Experiment 1



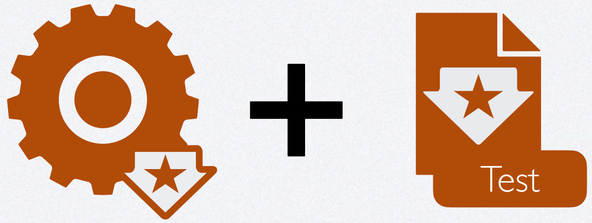

Experiment 2




Experiment 3




Experiment 4



# We found that models trained on high-star code tend to generalize better.


| | Model | Test Set | Accuracy | VarCLR |
|---|---|---|---|---|
| 1 | High Star | High Star | 33.6% | 0.708 |
| 2 | High Star | Low Star | 36.6% | 0.719 |
| 3 | Low Star | High Star | 15.6% | 0.622 |
| 4 | Low Star | Low Star | 39.6% | 0.734 |

# We found that models trained on high-star code tend to generalize better.


| | Model | Test Set | Accuracy | VarCLR |
|---|---|---|---|---|
| 1 | High Star | High Star | 33.6% | 0.708 |
| 2 | High Star | Low Star | 36.6% | 0.719 |
| 3 | Low Star | High Star | 15.6% | 0.622 |
| 4 | Low Star | Low Star | 39.6% | 0.734 |