

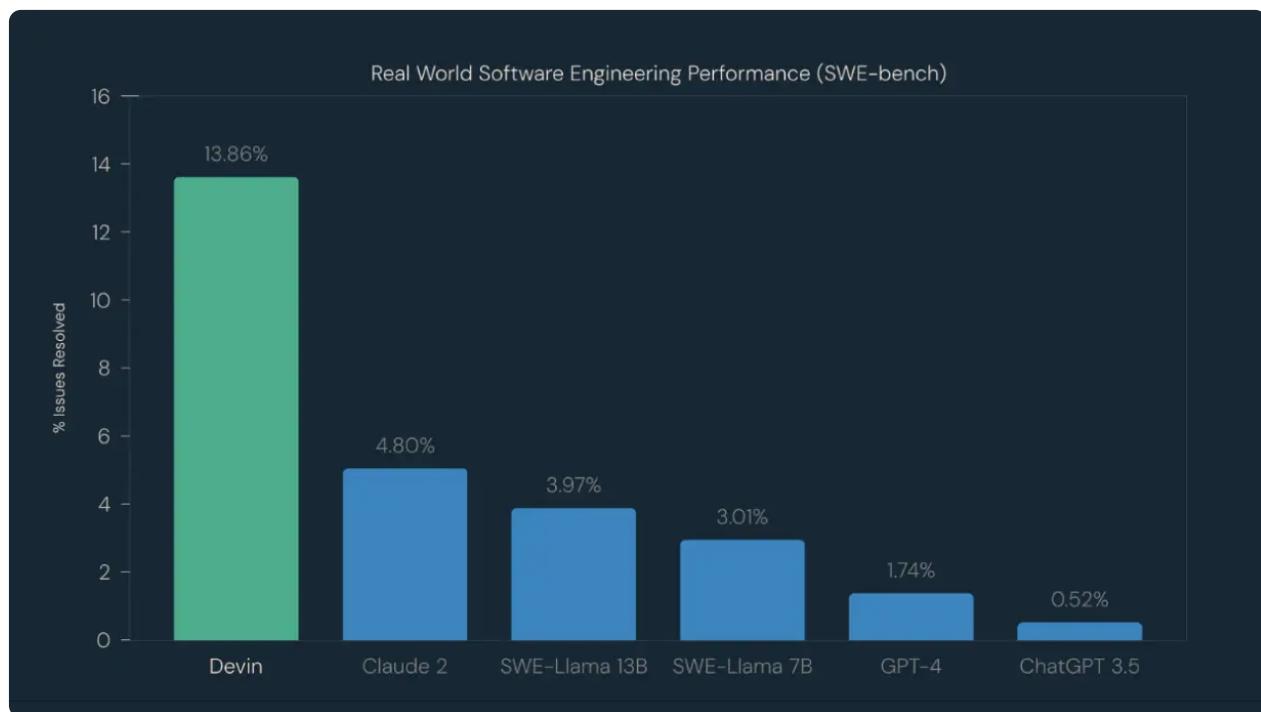
[home](#) • [blog](#)[/blog](#)

SWE-bench technical report

March 15, 2024 • by The Cognition Team

One of our goals at Cognition is to enable Devin, an AI agent specializing in software development, to contribute code successfully to large, complex codebases.

[announcements](#) • [benchmarking](#) • [reports](#)



To evaluate Devin, we turn to ***SWE-bench***, an automated benchmark for software engineering systems consisting of GitHub issues and pull requests. We believe SWE-bench is a great choice because it deterministically evaluates (via unit tests) a system's ability to solve issues in real world codebases, unlike benchmarks like ***HumanEval*** which are limited to standalone functions.

In SWE-bench, Devin successfully resolves 13.86%* of issues, far exceeding the previous highest unassisted baseline of 1.96%. Even when given the exact files to edit ("assisted"), the best previous model only resolves 4.80% of issues.

We provide our evaluation harness and Devin's code edits at <https://github.com/CognitionAI/devin-swebench-results>.

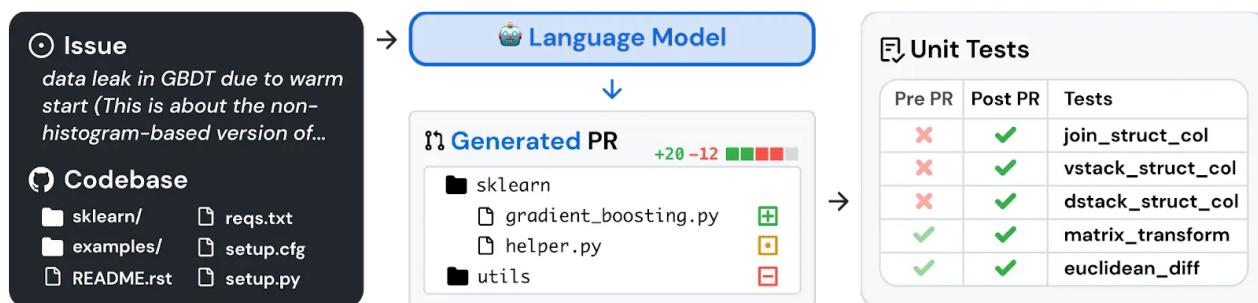
Background

SWE-bench is a dataset of 2,294 issues and pull requests scraped from popular open source Python repositories on GitHub. Its goal is to test a system's ability to write real-world code.

Each SWE-bench instance consists of a GitHub issue and the pull request which resolved it. The pull request must include a unit

test which fails before the code change and passes after (called a “fail to pass” test). The diff is split into two parts, patch and test_patch, which contain the code changes and the test changes, respectively.

Then the system being evaluated is asked to generate a diff given the GitHub issue description and the repository (at the time of the issue). The example is considered successful if all of the unit tests pass after patching the edit.



Source: swebench.com

In SWE-bench, LLMs are either given the set of correct files to edit (“assisted”); or a separate system retrieves the files to edit based on similarity to the issue text (“unassisted”). As an agent, Devin does not receive any list of files and instead navigates files on its own, which is more comparable to the “unassisted” LLM.

Properly solving SWE-bench examples is challenging. The harder PRs require changing tens of files, maintaining backwards compatibility, and/or doing a lot of complex reasoning. Even when assisted, the best LLMs only achieve a 4.80% success rate.

Methodology

We adapt SWE-bench to evaluate agents, a more general setting than the original eval for LLMs.

Setup

- We run the agent end to end using a standardized prompt that asks it to edit code given only the GitHub issue description. We do not give the agent any other user input during the run.

- The repo is cloned in the agent's environment. We only keep the base commit and its ancestors in the git history to prevent information leakage to the agent. Notably, we remove the git remote so that git pull does not work.
- We set up the Python conda environment before the test starts.
- We limit Devin to 45 minutes of runtime, as unlike most agents, it has the capability to run indefinitely. It can choose to terminate earlier if it wants.

Eval

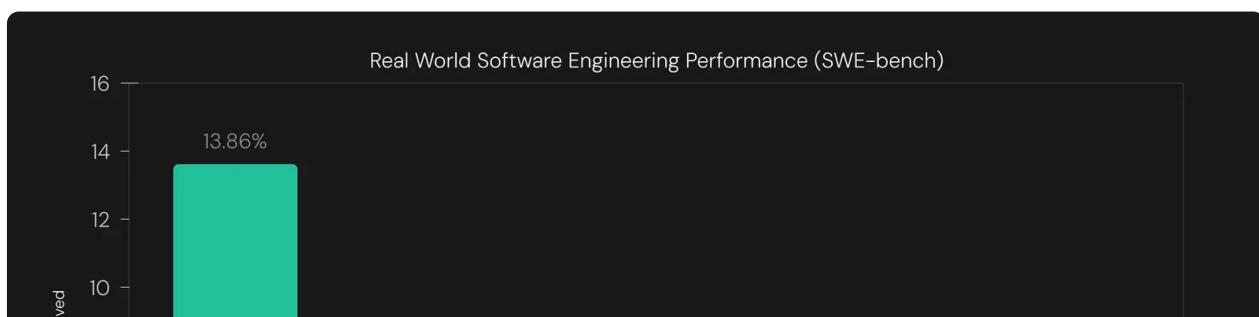
- Once the agent's run exits, we reset all of the test files to the original state, in case the agent modified the tests. We take all other diffs in the file system and extract them as a patch. To determine which files are test files, we take the set of all files that were modified in the test patch.
- We apply the agent's patch to the repo, followed by the test patch.
- We run the eval command provided by SWE-bench and check whether all the tests pass.

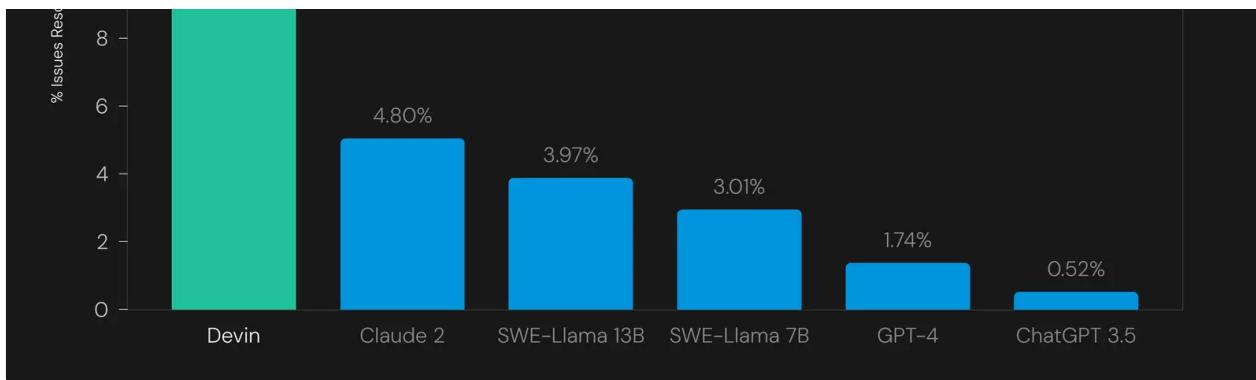
You can find code for our adapted eval harness at <https://github.com/CognitionAI/devin-swebench-results>.

Results

We evaluated Devin on a randomly chosen 25% of the SWE-benchmark test set (570 out of the 2,294). This was done to reduce the time it takes for the benchmark to finish, the same strategy the authors used in the original paper.

Devin successfully resolved 79 of the 570 issues, giving a 13.86% success rate. This is significantly higher than even the previous best assisted system (Claude 2) of 4.80%.





The baselines in this plot are evaluated in the “assisted” setting, where the model is provided with the exact file it needs to edit. Baselines perform worse in the “unassisted” setting, where a separate retrieval system selects the files for the LLM to edit (the best model is Claude 2 + BM25 retrieval with 1.96%).

Because neither unassisted nor assisted is strictly comparable to the agent setting, where Devin is given the entire repo and can navigate the files freely, we choose the stronger numbers for the baseline comparison. We believe that running an agent end to end is the more natural setting for SWE-bench, as it’s more similar to real-world software development. We anticipate more agent results in this setting going forward.

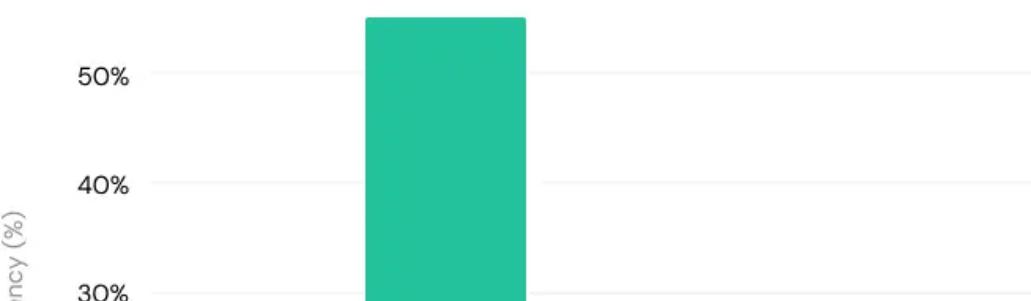
Analysis

Multi-step planning

Devin can execute multi-step plans to receive feedback from the environment. 72% of passing tests take over 10 minutes to complete, suggesting that the ability to iterate helps Devin succeed.

Histogram of Times for Passed Tests

Passed Tests





Qualitative examples

We provide some qualitative analysis of Devin's results. Recall that Devin is given just the issue description and the cloned repository as input.

Example 1: scikit-learn__scikit-learn-10870

Description

In Gaussian mixtures, when `n_init` is set to any value greater than 1, the `lower_bound_` is not the max lower bound across all initializations, but just the lower bound of the last initialization.

The bug can be fixed by adding the following line just before `return self` in `BaseMixture.fit()`:

```
self.lower_bound_ = max_lower_bound
```

The test that should have caught this bug is `test_init()` in `mixture/tests/test_gaussian_mixture.py`, but it just does a single test, so it had a 50% chance of missing the issue. It should be updated to try many random states.

Steps/Code to Reproduce

```
import numpy as np
from sklearn.mixture import GaussianMixture

X = np.random.rand(1000, 10)
for random_state in range(100):
    gm1 = GaussianMixture(n_components=2, n_init=1, random_state=random_state).fit(X)
    gm2 = GaussianMixture(n_components=2, n_init=10, random_state=random_state).fit(X)
    assert gm2.lower_bound_ > gm1.lower_bound_, random_state
```

Expected Results

No error.

Actual Results

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
AssertionError: 4
```

Devin is initially thrown off by the description, and adds `self.lower_bound_ = max_lower_bound` before return self literally as described. This is actually incorrect, as the variable isn't defined yet.

```

171
172     def fit(self, X, y=None):
173         """Estimate model parameters with the EM algori
174
175         The method fits the model `n_init` times and se
176         which the model has the largest likelihood or l
177         trial, the method iterates between E-step and M
178         times until the change of likelihood or lower b
179         `tol`, otherwise, a `ConvergenceWarning` is rai
180
181         Parameters
182
183             X : array-like, shape (n_samples, n_features)
184                 List of n_features-dimensional data points.
185                 corresponds to a single data point.
186
187         Returns
188
189             self
190
191             """
192             self.fit_predict(X, y)
193
194         return self

```

```

171
172     def fit(self, X, y=None):
173         """Estimate model parameters with the EM algori
174
175         The method fits the model `n_init` times and se
176         which the model has the largest likelihood or l
177         trial, the method iterates between E-step and M
178         times until the change of likelihood or lower_b
179         `tol`, otherwise, a `ConvergenceWarning` is rai
180
181         Parameters
182
183             X : array-like, shape (n_samples, n_features)
184                 List of n_features-dimensional data points.
185                 corresponds to a single data point.
186
187         Returns
188
189             self
190
191             """
192             self.fit_predict(X, y)
192+             self.lower_bound_ = max_lower_bound
193
194         return self

```

As test code provided in the issue description, Devin then updates the test file:

```

990
991 @ignore_warnings(category=ConvergenceWarning)
992 def test_init():
993     # We check that by increasing the n_init number we
994
994     random_state = 0
995     rand_data = RandomData(np.random.RandomState(random_
996     n_components = rand_data.n_components
997     X = rand_data.X['full']
998
999     gmm1 = GaussianMixture(n_components=n_components, n
1000         max_iter=1, random_state=rand_
1001     gmm2 = GaussianMixture(n_components=n_components, n
1002         max_iter=1, random_state=rand_
1003
1004     assert_greater(gmm2.lower_bound_, gmm1.lower_bound_

```

```

990
991 @ignore_warnings(category=ConvergenceWarning)
992 def test_init():
993     # Test that GaussianMixture with n_init > 1 indeed
994     # max lower bound across all initializations.
994+
995     random_state = 0
996     rand_data = RandomData(np.random.RandomState(random_
997     n_components = rand_data.n_components
998     X = rand_data.X['full']
999
1000     for random_state in range(10): # Test across multi
1001         gmm1 = GaussianMixture(n_components=n_component
1002             max_iter=1, random_state=rando
1003         gmm2 = GaussianMixture(n_components=n_component
1004             max_iter=1, random_state=rando
1005     assert gmm2.lower_bound_ >= gmm1.lower_bound_,

```

But upon running the tests and receiving errors, Devin corrects the file:

```

190
191     """
191     self.fit_predict(X, y)
192     self.lower_bound_ = max_lower_bound
193
194         return self

```

```

190
191     """
191     self.fit_predict(X, y)
192
193         return self

```

```

228
229     random_state = check_random_state(self.random_s
229
230     n_samples, _ = X.shape
230
231     for init in range(n_init):
231         self._print_verbose_msg_init_beg(init)
232
233         if do_init:
233             self._initialize_parameters(X, random_s
234             self.lower_bound_ = -np.infty
235

```

```

227
228     random_state = check_random_state(self.random_s
229
230     n_samples, _ = X.shape
230
231     for init in range(n_init):
231         self._print_verbose_msg_init_beg(init)
232
233         if do_init:
233             self._initialize_parameters(X, random_s
234             self.lower_bound_ = -np.infty
235

```

```
237
238     for n_iter in range(1, self.max_iter + 1):
239         prev_lower_bound = self.lower_bound_
240
241         log_prob_norm, log_resp = self._e_step(
242             self._m_step(X, log_resp)
243             self.lower_bound_ = self._compute_lower_
244                 log_resp, log_prob_norm)
245
246         change = self.lower_bound_ - prev_lower_
247         self._print_verbose_msg_iter_end(n_iter)
248
249         if abs(change) < self.tol:
250             self.converged_ = True
251             break
252
253         self._print_verbose_msg_init_end(self.lower_
254
255         if self.lower_bound_ > max_lower_bound:
256             max_lower_bound = self.lower_bound_
257             best_params = self._get_parameters()
258             best_n_iter = n_iter
259
260         if not self.converged_:
```

```
237
238     for n_iter in range(1, self.max_iter + 1):
239         prev_lower_bound = self.lower_bound_
240
241         log_prob_norm, log_resp = self._e_step(
242             self._m_step(X, log_resp)
243             self.lower_bound_ = self._compute_lower_
244                 log_resp, log_prob_norm)
245
246         change = self.lower_bound_ - prev_lower_
247         self._print_verbose_msg_iter_end(n_iter)
248
249         if abs(change) < self.tol:
250             self.converged_ = True
251             break
252
253         self._print_verbose_msg_init_end(self.lower_
254
255         if self.lower_bound_ > max_lower_bound:
256             max_lower_bound = self.lower_bound_
257             best_params = self._get_parameters()
258             best_n_iter = n_iter
259
260         if not self.converged_:
```

Upon this fix, Devin reruns the test to make it pass and successfully exits.

This example is interesting for a few reasons:

- Devin follows instructions from the original issue extremely closely, despite the inaccuracy. This indicates overalignment with the user's preferences.
 - Given the ability to run tests in its environment, Devin is able to correct its mistakes. It's crucial for software developers to be able to iterate, and agents should be able to do the same.

Example 2: ✓ django_django-10973

#30173	closed	Cleanup/optimization (fixed)	Opened 5 years ago
			Closed 5 years ago
Use subprocess.run and PGASSWORD for client in postgres backend			
Reported by:	Daniel Bowring	Owned by:	nobody
Component:	Database layer (models, ORM)	Version:	2.1
Severity:	Normal	Keywords:	
Cc:		Triage Stage:	Accepted
Has patch:	yes	Needs documentation:	no
Needs tests:	no	Patch needs improvement:	yes
Easy pickings:	no	UI/UX:	no
Pull Requests:	10973 merged		

Description

↳ `subprocess.run` was added in python 3.5 (which is the minimum version since Django 2.1). This function allows you to pass a custom environment for the subprocess.

Using this in `django.db.backends.postgres.client` to set `PGPASSWORD` simplifies the code and makes it more reliable.

Devin identifies the correct file `django/db/backends/postgresql/client.py`, and makes a complete edit:

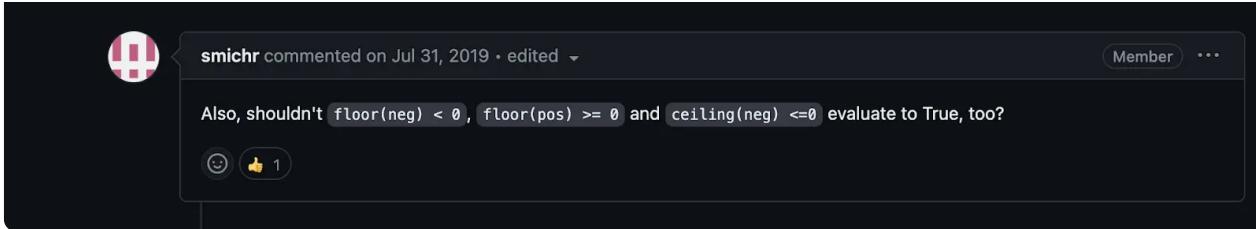
Here, Devin is able to modify a large chunk of code successfully. Many of the successful edits in SWE-bench consist of single line diffs, but Devin is able to handle several lines at once.

Example 3: X sympy_sympy-17313

This is a difficult task involving modifying a computer algebra system to correctly handle comparison operators on floor and ceiling objects in relation to values that can be specified to be positive or negative. It requires complex logical reasoning and multiple deduction steps.

ceiling(pos) > 0 should be true #17309

🕒 **Closed** smichr opened this issue on Jul 31, 2019 · 1 comment · Fixed by #17313



Devin misses the correct class to edit, editing the `frac` class rather than the `floor` class and `ceiling` class. On top of that Devin only edits one of the comparison operators, `__gt__`, when `__lt__`, `__le__`, and `__ge__` need to be modified as well. This edit is quite far from being correct.

```
388     other = _sympify(other)
389     # Check if other <= 0
390     if other.is_extended_nonpositive:
391         return S.true
392     # Check if other >= 1
393     res = self._value_one_or_more(other)
394     if res is not None:
395         return not(res)
396     # Check if other == 1
397     if other.is_extended_negative:
398         return S.true
399
400     return Gt(self, other, evaluate=False)
401
402     def __gt__(self, other):
403         if self.is_extended_real:
404             other = _sympify(other)
405             # Check if other < 0
406             res = self._value_one_or_more(other)
407             if res is not None:
408                 return not(res)
409             # Check if other >= 1
410             if other.is_extended_negative:
411                 return S.true
412
413             return Gt(self, other, evaluate=False)
414
415     def __le__(self, other):
416         if self.is_extended_real:
417             other = _sympify(other)
418             # Check if other < 0
419             if other.is_extended_negative:
420                 return S.false
421             # Check if other ~ -T
422             if other.is_extended_positive:
423                 return S.false
424
425             return Lt(self, other, evaluate=False)
426
427     def __ge__(self, other):
428         if self.is_extended_real:
429             other = _sympify(other)
430             # Check if other <= 0
431             if other.is_extended_nonpositive:
432                 return S.true
433             # Check if other >= 1
434             res = self._value_one_or_more(other)
435             if res is not None:
436                 return not(res)
437             # Check if other == 1
438             if other.is_extended_negative:
439                 return S.true
440
441             return Ge(self, other, evaluate=False)
442
443     def __gt__(self, other):
444         if self.is_extended_real:
445             other = _sympify(other)
446             # Check if other is a positive number
447             if other.is_positive:
448                 # If other is positive, ceiling of any number is greater than 0
449                 return S.true
450
451             # Check if other is non-positive
452             if other.is_nonpositive:
453                 # If other is non-positive, ceiling of any number is not greater
454                 return S.false
455
456             # If other is not a number, return unevaluated
457             return Gt(self, other, evaluate=False)
458
459     def __le__(self, other):
460         if self.is_extended_real:
461             other = _sympify(other)
462             # Check if other < 0
463             if other.is_extended_negative:
464                 return S.false
465             # Check if other ~ -T
466             if other.is_extended_positive:
467                 return S.false
468
469             return Lt(self, other, evaluate=False)
470
471     def __ge__(self, other):
472         if self.is_extended_real:
473             other = _sympify(other)
474             # Check if other <= 0
475             if other.is_extended_nonpositive:
476                 return S.true
477             # Check if other >= 1
478             res = self._value_one_or_more(other)
479             if res is not None:
480                 return not(res)
481             # Check if other == 1
482             if other.is_extended_negative:
483                 return S.true
484
485             return Ge(self, other, evaluate=False)
```

The correct diff can be found here:

<https://github.com/sympy/sympy/pull/17313/files>. The diff is quite complex with a lot of edge case handling and a large number of unit tests, and requires a deep understanding of the sympy codebase. (Note that every single test must pass in order to pass the SWE-bench instance.)

Example 4: X scikit-learn__scikit-learn-10774

This task involves adding additional return option functionality to all of the datasets in the repo. Devin is able to successfully make this edit for several of the datasets; an example is shown below.

```
66     data_home : optional, default: None
67         Specify another download and cache folder for the datasets. By default
68         all scikit-learn data is stored in '~/scikit_learn_data' subfolders.
69
70     download_if_missing : optional, default=True
71         If False, raise a IOError if the data is not locally available
72         instead of trying to download the data from the source site.
73
74
75     Returns
76     -----
77     dataset : dict-like object with the following attributes:
78
79     dataset.data : ndarray, shape [20640, 8]
80         Each row corresponding to the 8 feature values in order.
81
82     dataset.target : numpy array of shape (20640,)
83         Each value corresponds to the average house value in units of 100,000.
84
85     dataset.feature_names : array of length 8
86         Array of ordered feature names used in the dataset.
87
88     dataset.DESCR : string
89         Description of the California housing dataset.
90
91
92     Notes
93     -----
94
95
96     data_home : optional, default: None
97         Specify another download and cache folder for the datasets. By default
98         all scikit-learn data is stored in '~/scikit_learn_data' subfolders.
99
100    download_if_missing : optional, default=True
101        If False, raise a IOError if the data is not locally available
102        instead of trying to download the data from the source site.
103
104    return_X_y : boolean, default=False.
105        If True, returns (data, target) instead of a Bunch object.
106
107    Returns
108    -----
109    Ilambharathi Kanniah, 10 years ago • DOC adding return type and improvin
110    dataset : dict-like object with the following attributes:
111
112    dataset.data : ndarray, shape [20640, 8]
113        Each row corresponding to the 8 feature values in order.
114
115    dataset.target : numpy array of shape (20640,)
116        Each value corresponds to the average house value in units of 100,000.
117
118    dataset.feature_names : array of length 8
119        Array of ordered feature names used in the dataset.
120
121    dataset.DESCR : string
122        Description of the California housing dataset.
123
124    (data, target) : tuple if 'return_X_y' is True
```

```
131
132     # target in units of 100,000
133     target = target / 100000.0
134
135     return Bunch(data=data,
136                    target=target,
137                    feature_names=feature_names,
138                    DESCRIPTOR=MODULE_DOCS)
139
140
141+    if return_X_y:
142+        return data, target
143+    else:
144+        return Bunch(data=data, target=target,
145+                                     feature_names=feature_names, DESCRIPTOR=MODULE_DOCS)
146
```

Devin manages to make a similar edit for the datasets `california_housing.py`, `covtype.py`, `kddcup99.py`, and `mldata.py` (which the original PR actually excluded). Unfortunately Devin misses two of the datasets, `lfw.py` and `rcv1.py`, so the tests ultimately fail. We intend to improve Devin's capabilities for editing multiple files.

Test-driven experiment

We run an additional experiment where we provide Devin with the final unit test(s) along with the problem statement. In this “test-driven development” setting, the successful pass rate increases to 23% out of 100 sampled tests. (Note that any changes to the test itself are erased before evaluation.)

This result is incomparable to other results from SWE-bench as the agent had access to the ground truth test patch. Nonetheless, test-driven development is a common pattern in software engineering, so this setting is a natural extension for SWE-bench. A human giving an agent a targeted test to pass is a natural way for human engineers and agents to collaborate, and we expect to see more test-driven agents in the future.

Example of issues which Devin newly solved with the test

-  `django==django-13321:`

Devin solved this issue by adding a print statement right before the function, then running the unit test, and then editing the file based on the print statement. The presence of the test case made it easy for Devin to debug.

```
128     # RemovedInDjango40Warning.
129     serialized = self.serializer().dumps(session_dict)
130     hash = self._hash(serialized)
131     return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
132
133     def _legacy_decode(self, session_data):
134         # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
135
136         encoded_data = base64.b64decode(session_data.encode('ascii'))
137         try:
138             # could produce ValueError if there is no ':'
139             hash, serialized = encoded_data.split(b':', 1)
140             expected_hash = self._hash(serialized)
141             if not constant_time_compare(hash.decode(), expected_hash):
142                 raise SuspiciousSession("Session data corrupted")
143             else:
144
145             return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
146
147             def _legacy_decode(self, session_data):
148                 # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
149                 print("Session data before decoding:", session_data.encode('ascii'))
150
151                 encoded_data = base64.b64decode(session_data.encode('ascii'))
152
153                 try:
154                     # could produce ValueError if there is no ':'
155                     hash, serialized = encoded_data.split(b':', 1)
156                     expected_hash = self._hash(serialized)
157
158                     return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
159
160                     def _legacy_decode(self, session_data):
161                         # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
162                         # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
163                         print("Session data before decoding:", session_data.encode('ascii'))
164
165                         encoded_data = base64.b64decode(session_data.encode('ascii'))
166
167                         try:
168                             # could produce ValueError if there is no ':'
169                             hash, serialized = encoded_data.split(b':', 1)
170                             expected_hash = self._hash(serialized)
171
172                             return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
173
174                             def _legacy_decode(self, session_data):
175                                 # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
176                                 # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
177                                 print("Session data before decoding:", session_data.encode('ascii'))
178
179                                 encoded_data = base64.b64decode(session_data.encode('ascii'))
180
181                                 try:
182                                     # could produce ValueError if there is no ':'
183                                     hash, serialized = encoded_data.split(b':', 1)
184                                     expected_hash = self._hash(serialized)
185
186                                     return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
187
188                                     def _legacy_decode(self, session_data):
189                                         # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
190                                         # RemovedInDjango40Warning: pre-Django 3.1 format will be invalid.
191                                         print("Session data before decoding:", session_data.encode('ascii'))
192
193                                         encoded_data = base64.b64decode(session_data.encode('ascii'))
194
195                                         try:
196                                             # could produce ValueError if there is no ':'
197                                             hash, serialized = encoded_data.split(b':', 1)
198                                             expected_hash = self._hash(serialized)
199
200                                             return base64.b64encode(hash.encode() + b':' + serialized).decode('ascii')
```

 [django_django-16983](#): The new unit test asserts that an exact error message is emitted: "The value of 'filter_horizontal[0]' cannot include [...]" . It's not possible to pass the test without already knowing the precise wording of the error. This highlights an issue with the benchmark and shows that it's not possible to get a perfect score without the test patches.

Caveats

Given the popularity of the open-source repos in the benchmark, Devin's underlying models will contain data from these repositories. However, the baselines we are comparing to (Claude, GPT, Llama, etc.) face similar data contamination issues.

In addition, effort needs to be put in to prevent agents from finding external information about these PRs and potentially copying the diff. During test setup, we remove the Github remote and all future commits from the repository so agents can't access those directly. Agents with internet access could potentially find external information through other methods; we've manually inspected Devin's successful runs to ensure this hasn't happened.

Going forward

OUR TEAM

Agents are still in their infancy, and there's a lot of room for improvement. At Cognition, we believe that agents will dramatically improve in the near future. We're excited to see progress on SWE-bench and new benchmarks for tasks such as data analysis, browsing for information, and more.

Help us push the frontier of reasoning and planning. [We're hiring!](#)

/join-us

Our team is small and talent-dense. Our founding team has 10 IOI gold medals and includes leaders and builders who have worked at the cutting edge of applied AI at companies like Cursor, Scale AI, Lunchclub, Modal, Google DeepMind, Waymo, and Nuro.

Building Devin is just the first step—our hardest challenges still lie ahead. If you're excited to solve some of the world's biggest problems and build AI that can reason, learn more about our team and apply to one of the roles below.

/open-positions

/general

General Application

San Francisco Bay Area • Full time

/engineering

Machine Learning Researcher

San Francisco Bay Area • Full time

Software Engineer

San Francisco Bay Area • Full time

Full Stack Design Engineer

San Francisco Bay Area • Full time

home • blog •

privacy policy

terms of service

website terms

data processing addendum

acceptable use policy