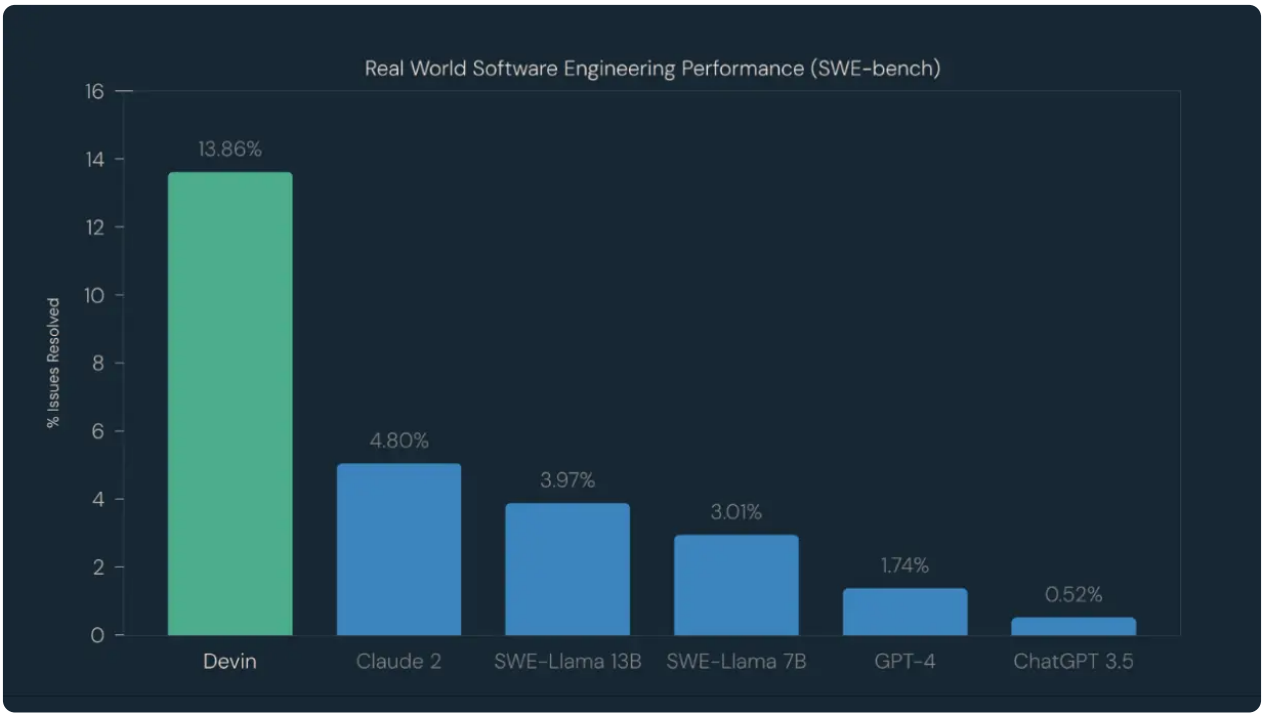
/blog

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***.

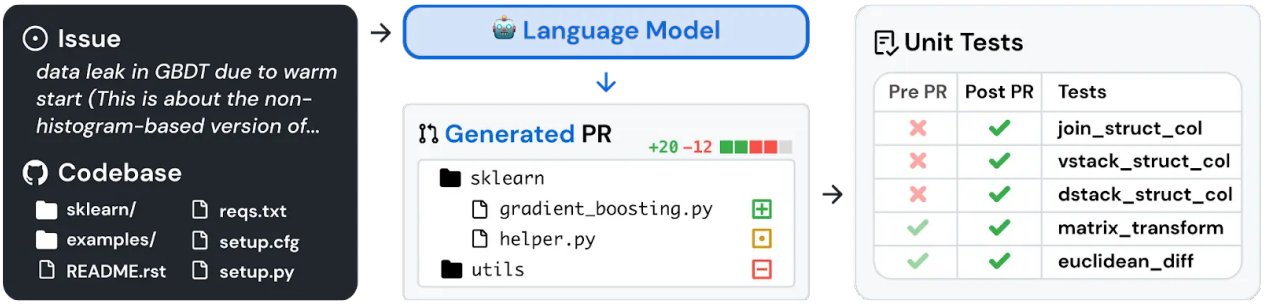
Background

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.

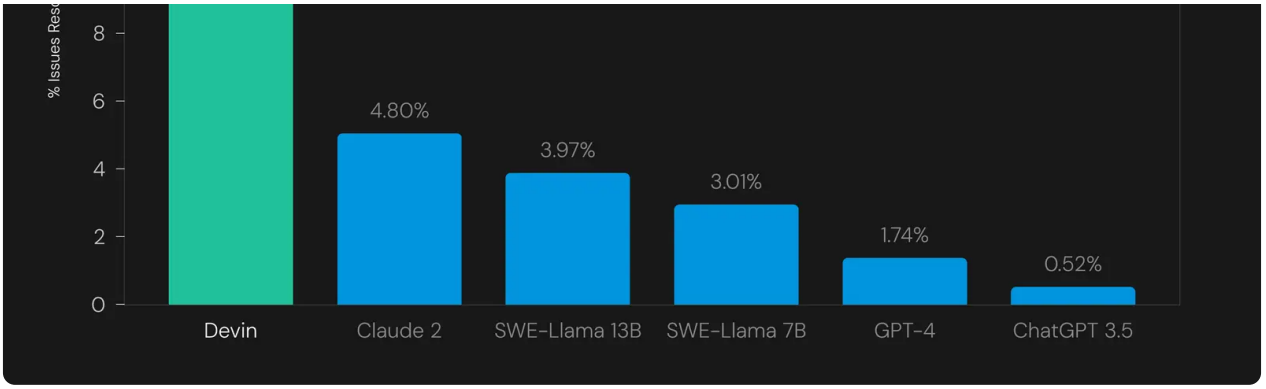
Eval

- 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%.



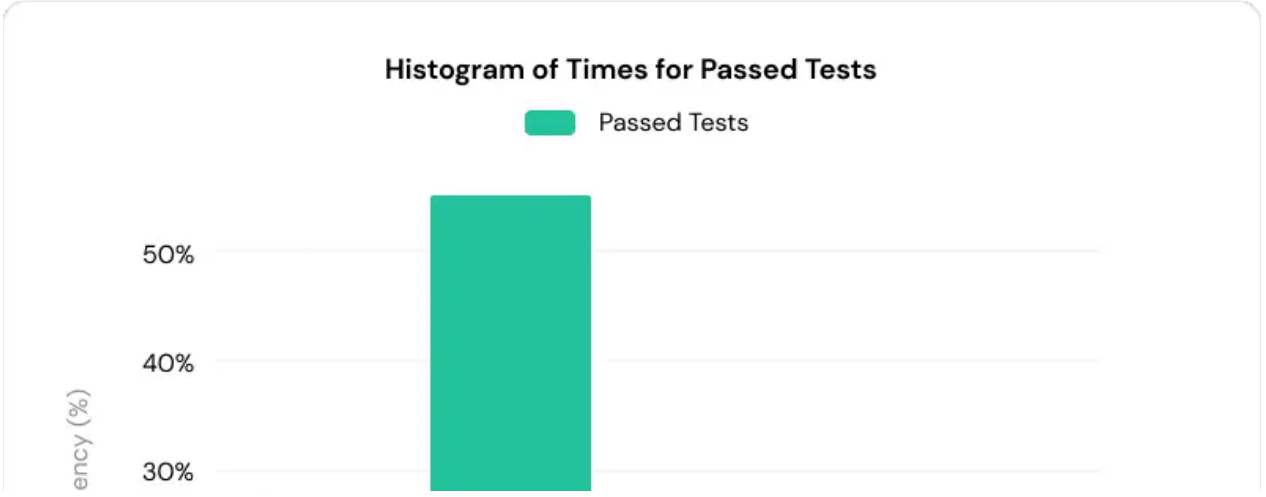
The baselines in this plot are evaluated in the "assisted" setting, where the model is provided with the exact file it needs to edit. Baselines perform worse in the "unassisted" setting, where a separate retrieval system selects the files for the LLM to edit (the best model is Claude 2 + BM25 retrieval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

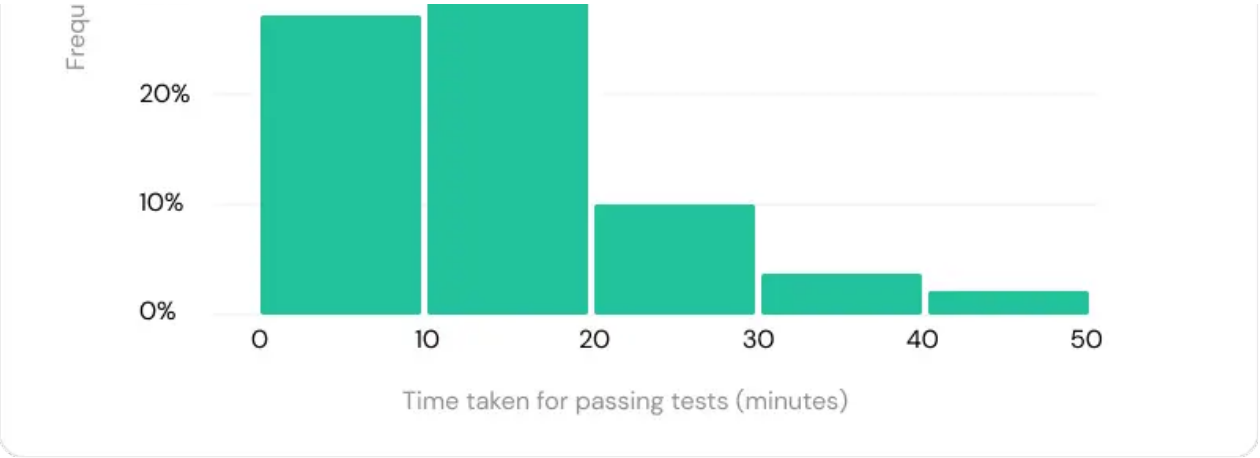
Because neither unassisted nor assisted is strictly comparable to the agent setting, where Devin is given the entire repo and can navigate the files freely, we choose the stronger numbers for the baseline comparison. We believe that running an agent end to end is the more natural setting for SWE-bench, 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.





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
```

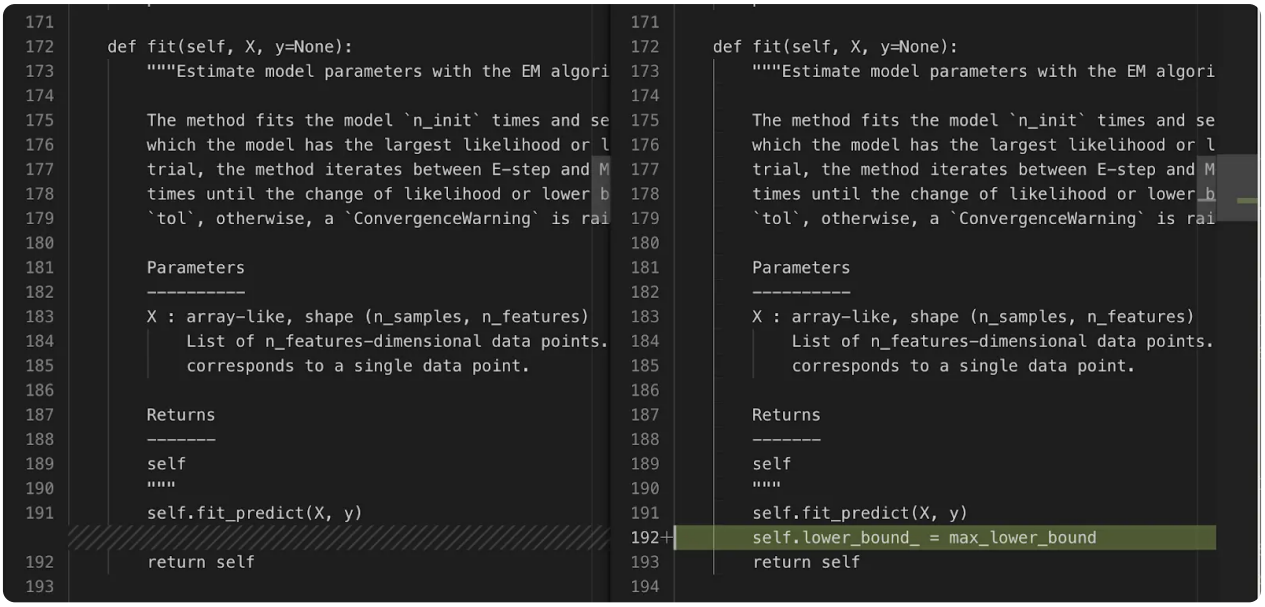
Expected Results

No error.

Actual Results

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

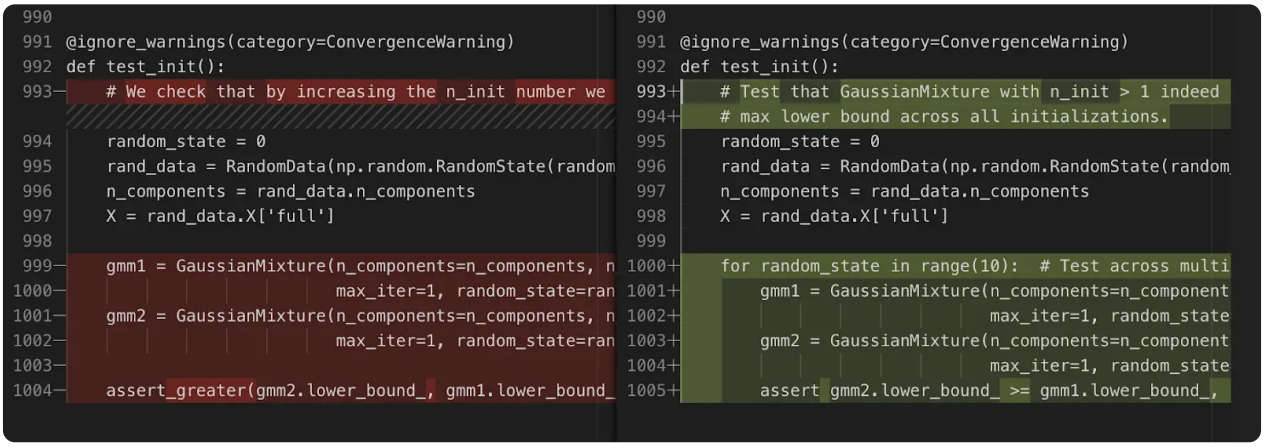
Devin is initially thrown off by the description, and adds self.lower_bound_ = max_lower_bound before return self literally as described. This is actually incorrect, as the variable isn't defined yet.



```
    def fit(self, X, y=None):
        """Estimate model parameters with the EM algori

        The method fits the model `n_init` times and se
        which the model has the largest likelihood or l
        trial, the method iterates between E-step and M
        times until the change of likelihood or lower b
        `tol`, otherwise, a `ConvergenceWarning` is rai

        Parameters
        ----------
        X : array-like, shape (n_samples, n_features)
            List of n_features-dimensional data points.
            corresponds to a single data point.

        Returns
        -------
        self
        """
        self.fit_predict(X, y)
        return self
```

```
    def fit(self, X, y=None):
        """Estimate model parameters with the EM algori

        The method fits the model `n_init` times and se
        which the model has the largest likelihood or l
        trial, the method iterates between E-step and M
        times until the change of likelihood or lower_b
        `tol`, otherwise, a `ConvergenceWarning` is rai

        Parameters
        ----------
        X : array-like, shape (n_samples, n_features)
            List of n_features-dimensional data points.
            corresponds to a single data point.

        Returns
        -------
        self
        """
        self.fit_predict(X, y)
        self.lower_bound_ = max_lower_bound
        return self
```

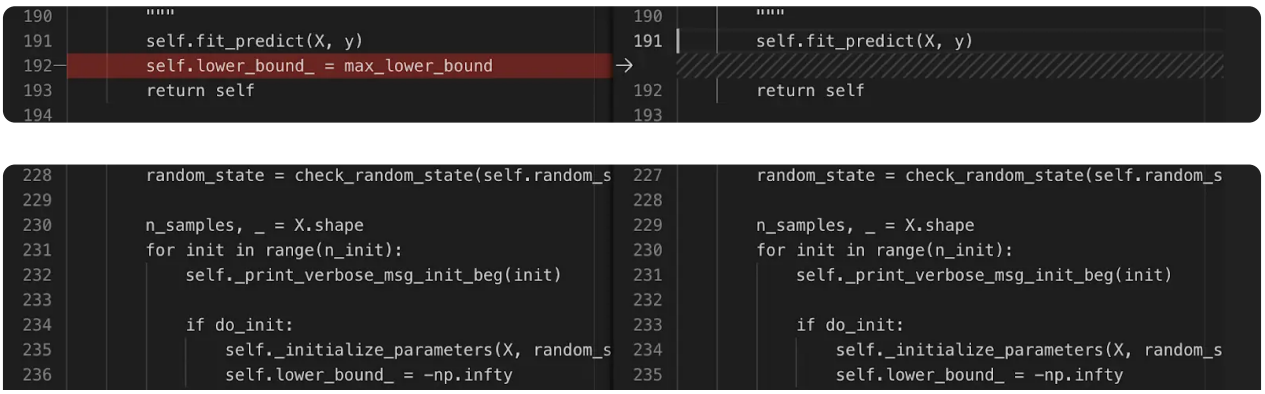
As test code provided in the issue description, Devin then updates the test file:



```
@ignore_warnings(category=ConvergenceWarning)
def test_init():
    # We check that by increasing the n_init number we
    random_state = 0
    rand_data = RandomData(np.random.RandomState(random
    n_components = rand_data.n_components
    X = rand_data.X['full']

    gmm1 = GaussianMixture(n_components=n_components, n
                           max_iter=1, random_state=ran
    gmm2 = GaussianMixture(n_components=n_components, n
                           max_iter=1, random_state=ran

    assert_greater(gmm2.lower_bound_, gmm1.lower_bound_
```

```
@ignore_warnings(category=ConvergenceWarning)
def test_init():
    # Test that GaussianMixture with n_init > 1 indeed
    # max lower bound across all initializations.
    random_state = 0
    rand_data = RandomData(np.random.RandomState(random
    n_components = rand_data.n_components
    X = rand_data.X['full']

    for random_state in range(10):  # Test across multi
        gmm1 = GaussianMixture(n_components=n_component
                               max_iter=1, random_state
        gmm2 = GaussianMixture(n_components=n_component
                               max_iter=1, random_state
        assert gmm2.lower_bound_ >= gmm1.lower_bound_,
```

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

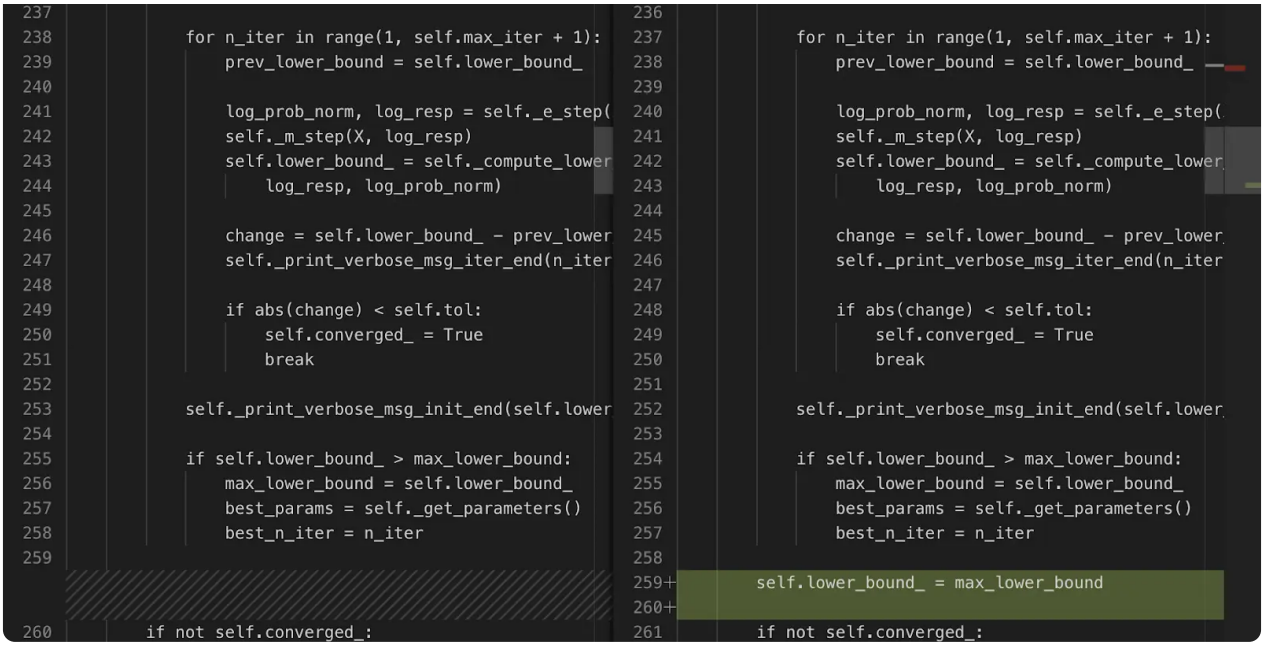


```
        """
        self.fit_predict(X, y)
        self.lower_bound_ = max_lower_bound
        return self
```

```
        """
        self.fit_predict(X, y)
        return self
```

```
        random_state = check_random_state(self.random_s

        n_samples, _ = X.shape
        for init in range(n_init):
            self._print_verbose_msg_init_beg(init)

            if do_init:
                self._initialize_parameters(X, random_s
                self.lower_bound_ = -np.infty
```

```
        random_state = check_random_state(self.random_s

        n_samples, _ = X.shape
        for init in range(n_init):
            self._print_verbose_msg_init_beg(init)

            if do_init:
                self._initialize_parameters(X, random_s
                self.lower_bound_ = -np.infty
```



```
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_:
```

```
236
237         for n_iter in range(1, self.max_iter + 1):
238             prev_lower_bound = self.lower_bound_
239
240             log_prob_norm, log_resp = self._e_step(
241             self._m_step(X, log_resp)
242             self.lower_bound_ = self._compute_lower
243                 log_resp, log_prob_norm)
244
245             change = self.lower_bound_ - prev_lower
246             self._print_verbose_msg_iter_end(n_iter
247
248             if abs(change) < self.tol:
249                 self.converged_ = True
250                 break
251
252         self._print_verbose_msg_init_end(self.lower
253
254         if self.lower_bound_ > max_lower_bound:
255             max_lower_bound = self.lower_bound_
256             best_params = self._get_parameters()
257             best_n_iter = n_iter
258
259+            self.lower_bound_ = max_lower_bound
260+
261         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 PGPASSWORD for client in postgres backend

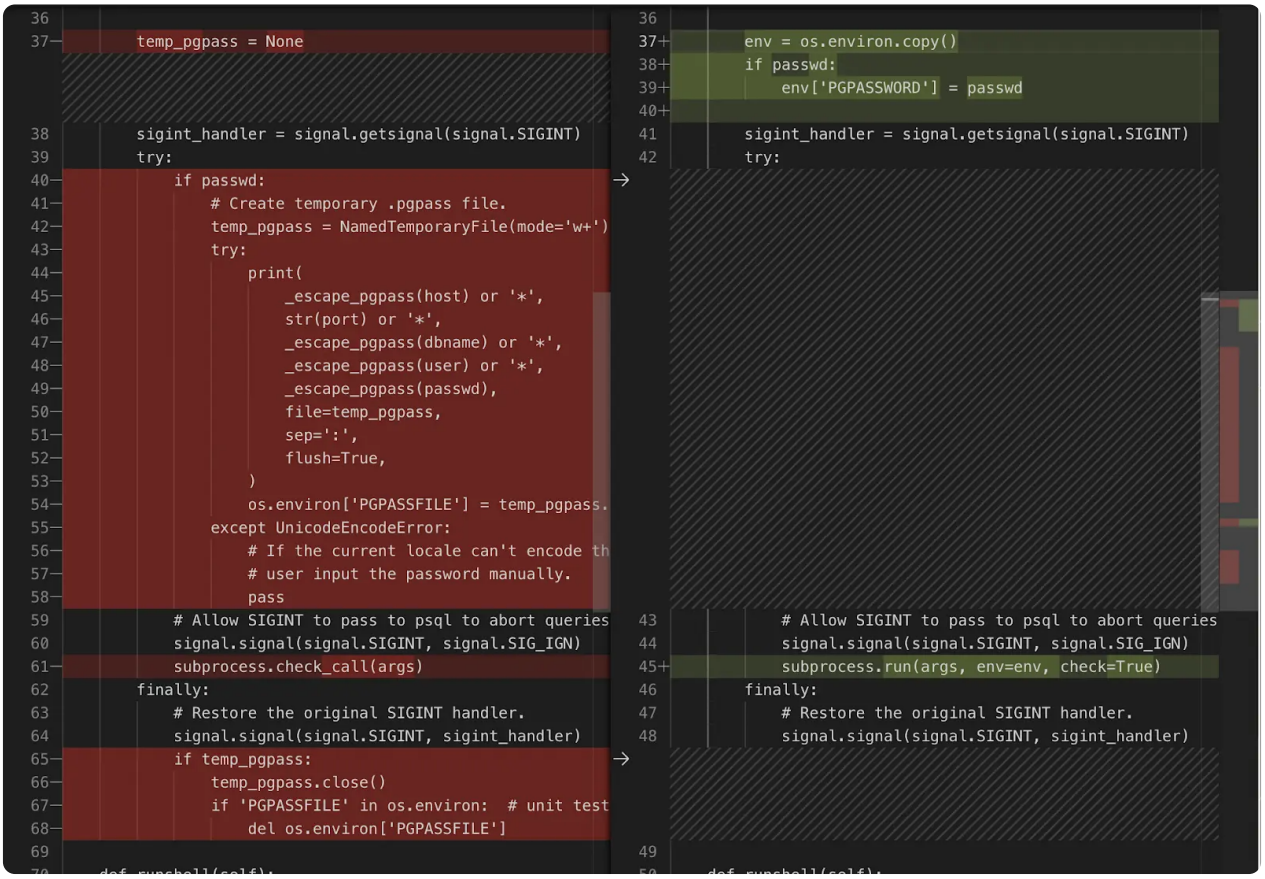
Reported by:	Daniel Bowring	Owned by:	nobody
Component:	Database layer (models, ORM)	Version:	2.1
Severity:	Normal	Keywords:	
Cc:		Triage Stage:	Accepted
Has patch:	yes	Needs documentation:	no
Needs tests:	no	Patch needs improvement:	yes
Easy pickings:	no	UI/UX:	no
Pull Requests:	~~10973~~ merged		

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.

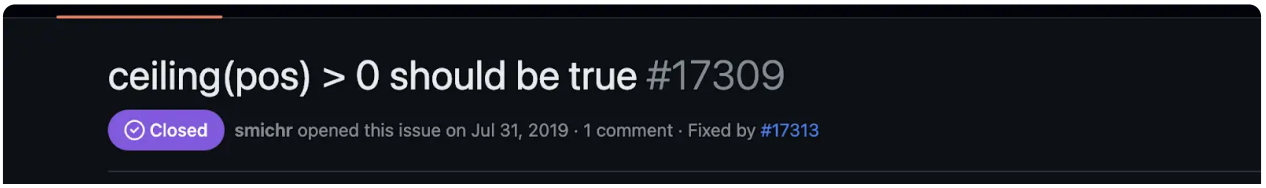
Devin identifies the correct file django/db/backends/postgresql/client.py, and makes a complete edit:

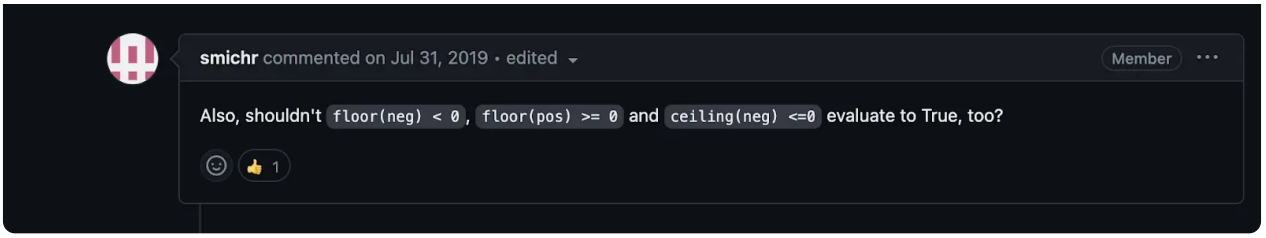


Here, Devin is able to modify a large chunk of code successfully. Many of the successful edits in SWE-bench consist of single line diffs, but Devin is able to handle several lines at once.

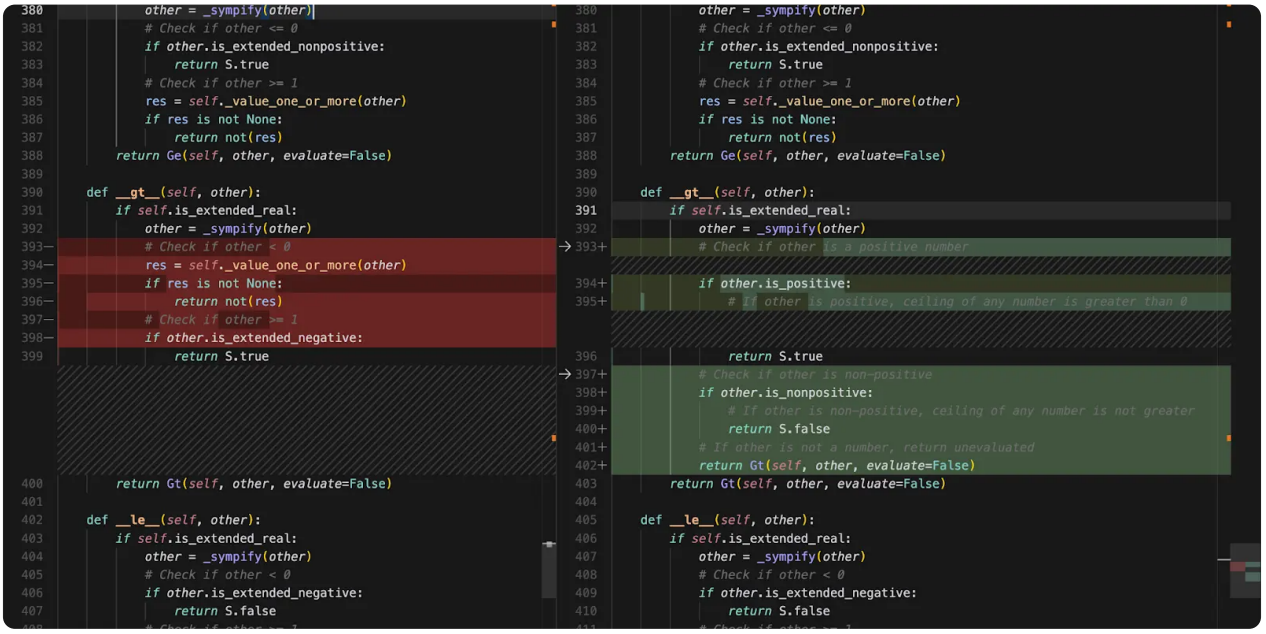
Example 3:* ✗ *sympy__sympy-17313

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.





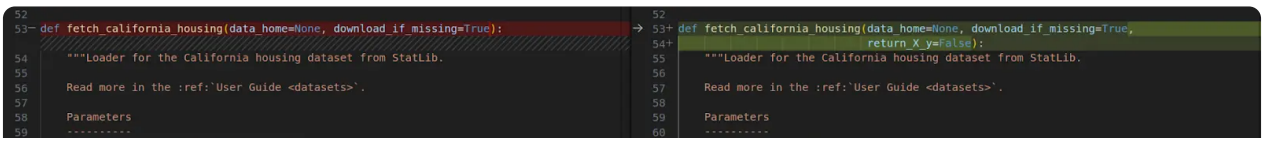
Devin misses the correct class to edit, editing the frac class rather than the floor class and ceiling class. On top of that Devin only edits one of the comparison operators, __gt__, when __lt__, __le__, and __ge__ need to be modified as well. This edit is quite far from being correct.

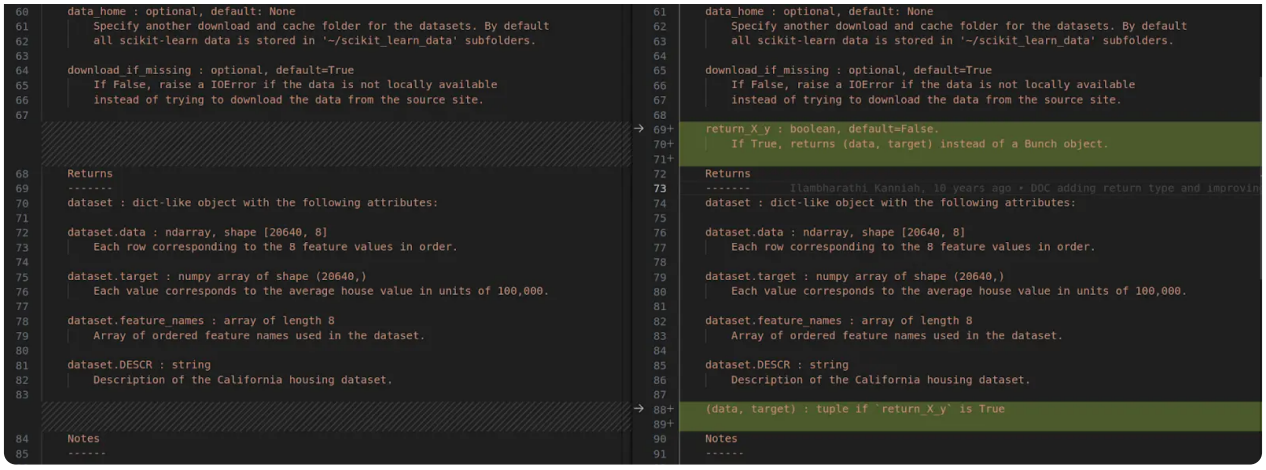


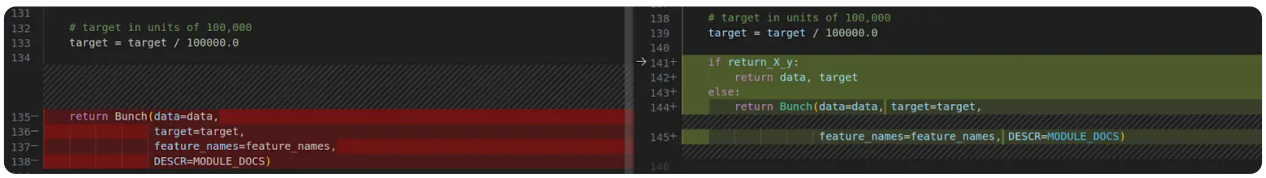
The correct diff can be found here: ***https://github.com/sympy/sympy/pull/17313/files***. The diff is quite complex with a lot of edge case handling and a large number of unit tests, and requires a deep understanding of the sympy codebase. (Note that every single test must pass in order to pass the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

Example 4: ✗ 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.







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.

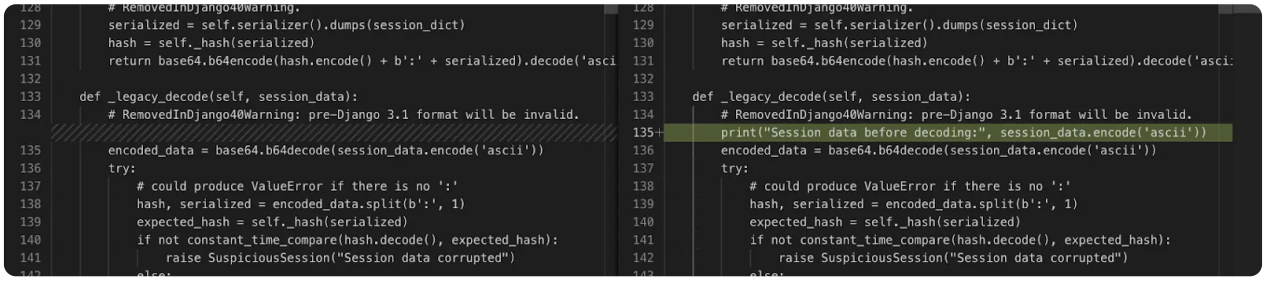
Test-driven experiment

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.



✅ ***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

Going forward

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