Instance ID: sympy__sympy-8273		
Gold Patch	**Aider (GPT4o & Claude 3 Opus)**	**AppMap Naive (GPT4o)**

`from sphinx.util.osutil import ensuredir, make_filename_from_project`
 	
`if self.config.man_make_section_directory:`
`  # Correct Fix: Properly use existing ensuredir utility for creating directories`
`  ensuredir(path.join(self.outdir, str(section)))`
`  targetname = '%s/%s.%s' % (section, name, section)`
`else:`
`  targetname = '%s.%s' % (name, section)`
	`from os import path, makedirs`
	
	`section_dir = path.join(self.outdir, f"man{section}")`
	`# Wrong Fix: Using makedirs() instead of the existing ensuredir() utility`
	`if not path.exists(section_dir):`
	`  makedirs(section_dir)`
	`targetname = path.join(section_dir, '%s.%s' % (name, section))`
		`for i, line in enumerate(result_lines):`
		`  # Wrong Fix: Manually handling directory creation instead of leveraging ensuredir()`
		`  if line.rstrip().endswith("_\\"):`
		`  result_lines[i] = line[:-2] + "_"`

Table 1: Model Limitations in Integrating with Existing Utilities - Comparing Gold Patch vs. Model-Generated Patches (sympy__sympy-8273). Green texts indicate the correct implementation in the gold patch, while red text highlights the flawed approaches in model-generated patches.

Instance ID: sympy__sympy-19542		
Gold Patch	**AutoCodeRover+GPT4o Patch**	**Agentless +GPT4o Patch**

`# Correct fix: Apply Knuth-Cohen bound with binomial coefficients and Euclidean norm for accurate estimation`
`from sympy import binomial`
`d = dup_degree(f)`
`delta = _ceil(d / 2)`
`delta2 = _ceil(delta / 2)`
`eucl_norm = K.sqrt(sum([cf**2 for cf in f]))`
`t1 = binomial(delta - 1, delta2)`
`t2 = binomial(delta - 1, delta2 - 1)`
`lc = K.abs(dup_LC(f, K))`
`bound = t1 * eucl_norm + t2 * lc`
`bound += dup_max_norm(f, K)`
`bound = _ceil(bound / 2) * 2`
`return bound`
	`def dup_zz_mignotte_bound(f, K):`
	`  a = dup_max_norm(f, K)`
	`  b = abs(dup_LC(f, K))`
	`  n = dup_degree(f)`
	`  # Wrong fix: Incorrect simplified formula that omits binomial coefficients and Euclidean norms`
	`  return ((K(n + 1) / 2) ** (n / 2)) * a * b`
	
	`def dmp_zz_mignotte_bound(f, u, K):`
	`  a = dmp_max_norm(f, u, K)`
	`  b = abs(dmp_ground_LC(f, u, K))`
	`  n = sum(dmp_degree_list(f, u))`
	`  # Wrong fix: Incorrect simplified formula that omits binomial coefficients and Euclidean norms`
	`  return ((K(n + 1) / 2) ** (n / 2)) * a * b`
		`a = dup_max_norm(f, K)`
		`b = abs(dup_LC(f, K))`
		`n = dup_degree(f)`
		`# Wrong fix: Retains original Mignotte formula without improving the estimation`
		`return 2**n * K.sqrt(K(n + 1)) * a * b`
		
		`def dmp_zz_mignotte_bound(f, u, K):`
		`  a = dmp_max_norm(f, u, K)`
		`  b = abs(dmp_ground_LC(f, u, K))`
		`  n = sum(dmp_degree_list(f, u))`
		`  # Wrong fix: Retains original Mignotte formula without improving the estimation`
		`  return 2**n * K.sqrt(K(n + 1)) * a * b`

Table 2: Model Limitations in Performance Implications - Comparing Gold Patch vs. Model-Generated Patches (sympy__sympy-19542). Green texts indicate the correct implementation in the gold patch, while red text highlights the flawed approaches in model-generated patches.

Instance ID: sympy__sympy-12454

Gold Patch

```
def _eval_is_upper_hessenberg(self):
    return all(self[i, j].is_zero
        for i in range(2, self.rows)
        for j in range(min(self.cols
    , (i - 1))))

def _eval_values(self):
    return [i for i in self if not i
    .is_zero]

# Correct Fix: Correctly limits the
    column index to prevent out-of-
    bounds errors
def is_upper(self):
    return all(self[i, j].is_zero
        for i in range(1, self.rows)
        for j in range(min(i,
    self.cols))

@property
def is_zero(self):
    pass
```

Swe-Agent+Claude 3 Opus Patch

```
def is_upper(self):
    return all(self[i, j].is_zero
        for i in range(1, self.rows)
        for j in range(min(i,
    self.cols))

# Wrong Fix: Incorrectly fixes only
    the is_upper() method and
    neglects
    _eval_is_upper_hessenberg().
@property
def is_zero(self):
    pass
```

Agentless+GPT4o

```
class MatrixProperties(
    MatrixRequired):
    return all(self[i, j].is_zero
        for i in range(1, self.
    rows)
        for j in range(min(i,
    self.cols))

# Wrong Fix: Incorrectly fixes
    only the is_upper() method
    and overlooks
    _eval_is_upper_hessenberg().
@property
def is_zero(self):
    pass
```

Table 3: Model Limitations in Error Handling - Comparing Gold Patch vs. Model-Generated Patches (sympy__sympy-12454). Green texts indicate the correct implementation in the gold patch, while red text highlights the flawed approaches in model-generated patches.

Instance ID: matplotlib__matplotlib-24970

Gold Patch

```
def __call__(self, X, alpha=None,
    bytes=False):
    if not xa.dtype.isnative:
        xa = xa.byteswap().
    newbyteorder()
    if xa.dtype.kind == "f":
        xa *= self.N
        xa[xa < 0] = -1
        xa[xa == self.N] = self.N
    - 1
        np.clip(xa, -1, self.N,
    out=xa)
        xa = xa.astype(int)
        # Correct:  Explicitly
    clips integer values to the
    valid range, ensuring
    cross-version compatibility.
        xa = np.clip(xa, 0, self.N
     - 1)
    xa[xa > self.N - 1] = self.
    _i_over
    xa[xa < 0] = self._i_under
    xa[mask_bad] = self._i_bad
```

Aider (GPT4o & Claude 3 Opus)

```
class Colormap:
    xa = xa.astype(int)
    xa = np.clip(xa, 0, self.N - 1)
    xa = np.clip(xa, -1, self.N)
    xa = np.clip(xa, -1, self.N)
    # Wrong Fix:  Redundant and
    inconsistent clipping that does
    not explicitly address NumPy's
    version-specific behavior.
    xa = np.clip(xa, -1, self.N)
    xa[xa > self.N - 1] = self.
    _i_over
    xa[xa < 0] = self._i_under
    xa[mask_bad] = self._i_bad
```

Moatless Tools (GPT4o1)

```
class Colormap:
    np.clip(xa, -1, self.N, out=
    xa)
    xa = xa.astype(int)
    # Simplifies the logic but
    removes explicit handling
    for over/under-range values,
    risking incorrect behavior.
    np.clip(xa, 0, self.N - 1,
    out=xa)
    xa[mask_bad] = self._i_bad
    lut = self._lut
```

Table 4: Model Limitations in Cross Version - Comparing Gold Patch vs. Model-Generated Patches (matplotlib__matplotlib-24970). Green texts indicate the correct implementation in the gold patch, while red text highlights the flawed approaches in model-generated patches.