AI ASSISTED CODING END LAB EXAM

N.PRANAY KUMAR

2403A52087

Batch – 04

SUBSET-05)

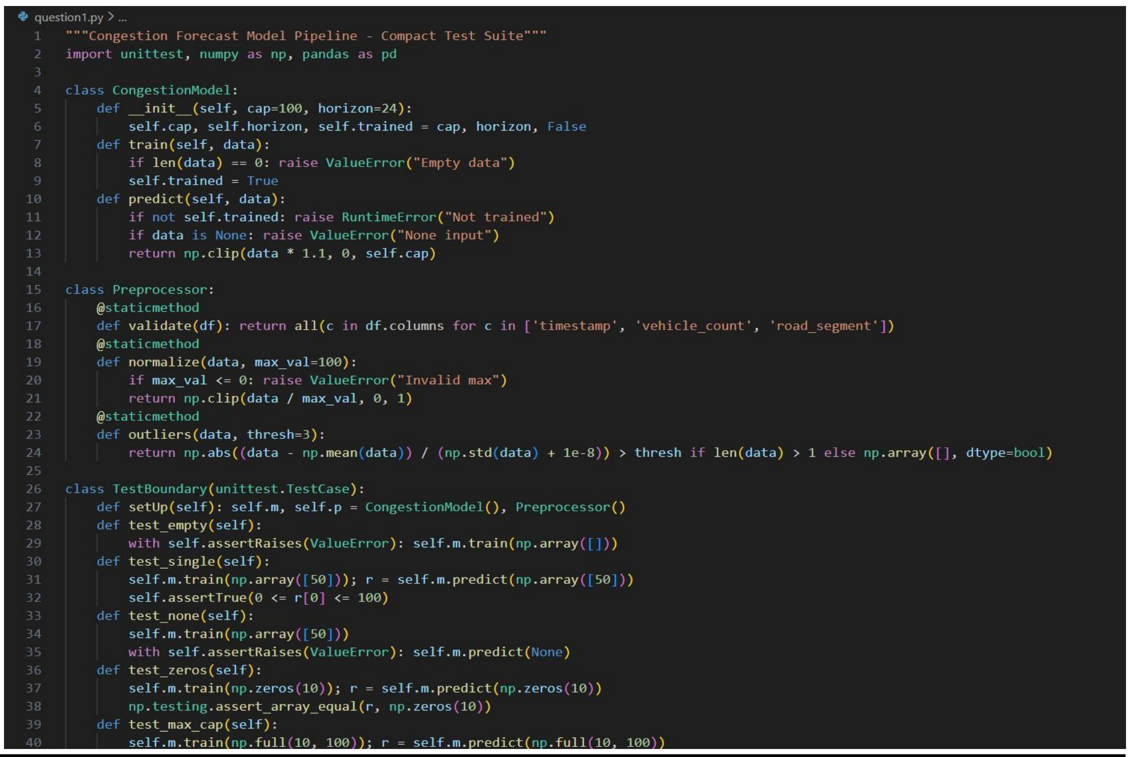
Q1) Create test cases for congestion forecast model pipeline.

• Task 1: Use AI to generate boundaries and stress tests.

• Task 2: Implement tests and ensure CI coverage.

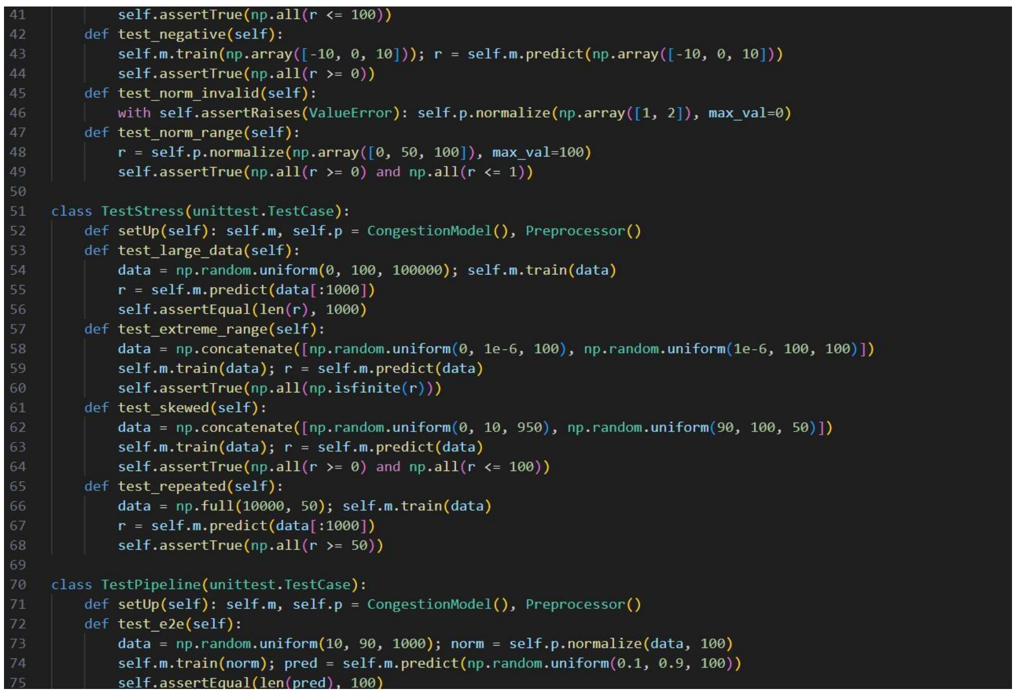
Prompt: Create test cases for congestion forecast model pipeline. Generate boundaries and stress tests. Implement tests and ensure CI coverage

Code:

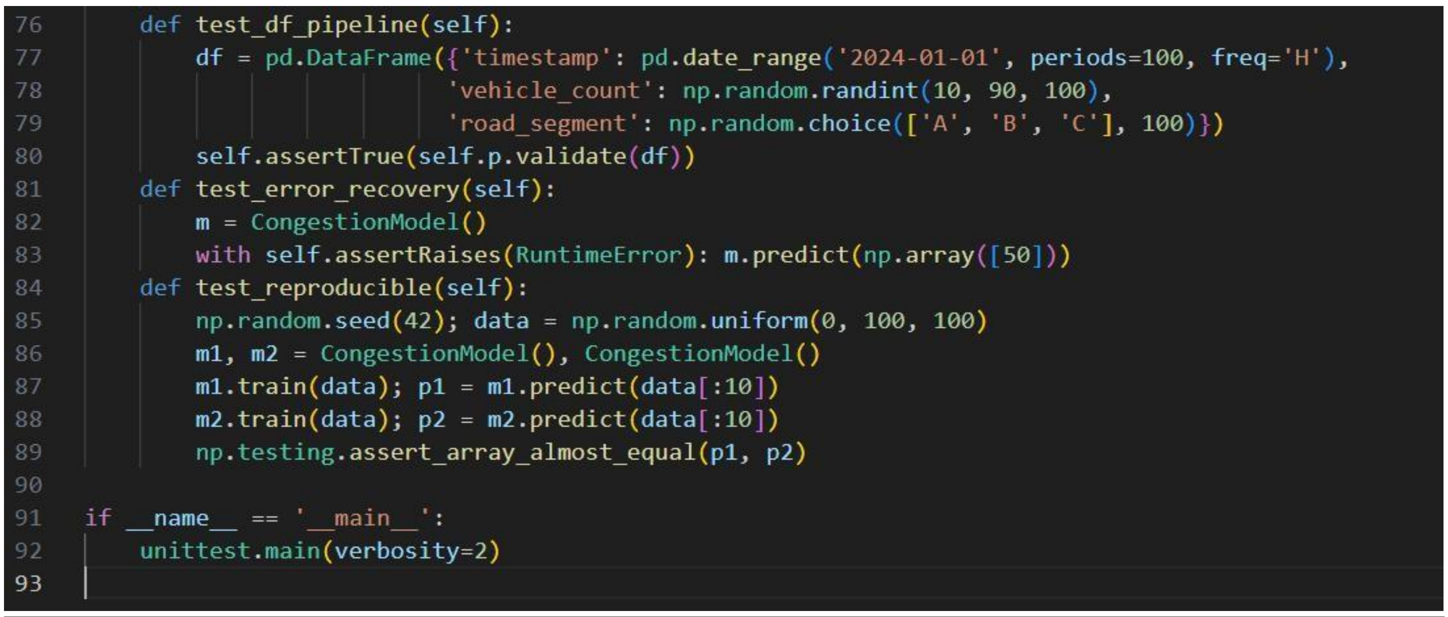


question1.py > ...

```python
"""Congestion Forecast Model Pipeline - Compact Test Suite"""
import unittest, numpy as np, pandas as pd

class CongestionModel:
    def __init__(self, cap=100, horizon=24):
        self.cap, self.horizon, self.trained = cap, horizon, False
    def train(self, data):
        if len(data) == 0: raise ValueError("Empty data")
        self.trained = True
    def predict(self, data):
        if not self.trained: raise RuntimeError("Not trained")
        if data is None: raise ValueError("None input")
        return np.clip(data * 1.1, 0, self.cap)

class Preprocessor:
    @staticmethod
    def validate(df): return all(c in df.columns for c in ['timestamp', 'vehicle_count', 'road_segment'])
    @staticmethod
    def normalize(data, max_val=100):
        if max_val <= 0: raise ValueError("Invalid max")
        return np.clip(data / max_val, 0, 1)
    @staticmethod
    def outliers(data, thresh=3):
        return np.abs((data - np.mean(data)) / (np.std(data) + 1e-8)) > thresh if len(data) > 1 else np.array([], dtype=bool)

class TestBoundary(unittest.TestCase):
    def setUp(self): self.m, self.p = CongestionModel(), Preprocessor()
    def test_empty(self):
        with self.assertRaises(ValueError): self.m.train(np.array([]))
    def test_single(self):
        self.m.train(np.array([50])); r = self.m.predict(np.array([50]))
        self.assertTrue(0 <= r[0] <= 100)
    def test_none(self):
        self.m.train(np.array([50]))
        with self.assertRaises(ValueError): self.m.predict(None)
    def test_zeros(self):
        self.m.train(np.zeros(10)); r = self.m.predict(np.zeros(10))
        np.testing.assert_array_equal(r, np.zeros(10))
    def test_max_cap(self):
        self.m.train(np.full(10, 100)); r = self.m.predict(np.full(10, 100))
```

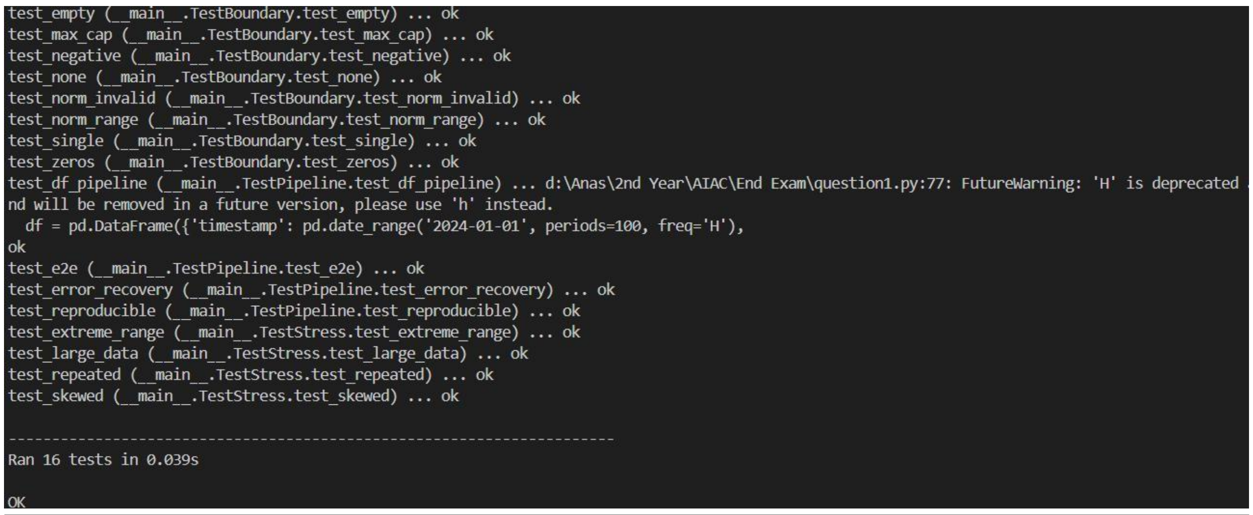


```python
        self.assertTrue(np.all(r <= 100))
    def test_negative(self):
        self.m.train(np.array([-10, 0, 10])); r = self.m.predict(np.array([-10, 0, 10]))
        self.assertTrue(np.all(r >= 0))
    def test_norm_invalid(self):
        with self.assertRaises(ValueError): self.p.normalize(np.array([1, 2]), max_val=0)
    def test_norm_range(self):
        r = self.p.normalize(np.array([0, 50, 100]), max_val=100)
        self.assertTrue(np.all(r >= 0) and np.all(r <= 1))

class TestStress(unittest.TestCase):
    def setUp(self): self.m, self.p = CongestionModel(), Preprocessor()
    def test_large_data(self):
        data = np.random.uniform(0, 100, 100000); self.m.train(data)
        r = self.m.predict(data[:1000])
        self.assertEqual(len(r), 1000)
    def test_extreme_range(self):
        data = np.concatenate([np.random.uniform(0, 1e-6, 100), np.random.uniform(1e-6, 100, 100)])
        self.m.train(data); r = self.m.predict(data)
        self.assertTrue(np.all(np.isfinite(r)))
    def test_skewed(self):
        data = np.concatenate([np.random.uniform(0, 10, 950), np.random.uniform(90, 100, 50)])
        self.m.train(data); r = self.m.predict(data)
        self.assertTrue(np.all(r >= 0) and np.all(r <= 100))
    def test_repeated(self):
        data = np.full(10000, 50); self.m.train(data)
        r = self.m.predict(data[:1000])
        self.assertTrue(np.all(r >= 50))

class TestPipeline(unittest.TestCase):
    def setUp(self): self.m, self.p = CongestionModel(), Preprocessor()
    def test_e2e(self):
        data = np.random.uniform(10, 90, 1000); norm = self.p.normalize(data, 100)
        self.m.train(norm); pred = self.m.predict(np.random.uniform(0.1, 0.9, 100))
        self.assertEqual(len(pred), 100)
```



```python
    def test_df_pipeline(self):
        df = pd.DataFrame({'timestamp': pd.date_range('2024-01-01', periods=100, freq='H'),
                           'vehicle_count': np.random.randint(10, 90, 100),
                           'road_segment': np.random.choice(['A', 'B', 'C'], 100)})
        self.assertTrue(self.p.validate(df))
    def test_error_recovery(self):
        m = CongestionModel()
        with self.assertRaises(RuntimeError): m.predict(np.array([50]))
    def test_reproducible(self):
        np.random.seed(42); data = np.random.uniform(0, 100, 100)
        m1, m2 = CongestionModel(), CongestionModel()
        m1.train(data); p1 = m1.predict(data[:10])
        m2.train(data); p2 = m2.predict(data[:10])
        np.testing.assert_array_almost_equal(p1, p2)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Output:



```
test_empty (__main__.TestBoundary.test_empty) ... ok
test_max_cap (__main__.TestBoundary.test_max_cap) ... ok
test_negative (__main__.TestBoundary.test_negative) ... ok
test_none (__main__.TestBoundary.test_none) ... ok
test_norm_invalid (__main__.TestBoundary.test_norm_invalid) ... ok
test_norm_range (__main__.TestBoundary.test_norm_range) ... ok
test_single (__main__.TestBoundary.test_single) ... ok
test_zeros (__main__.TestBoundary.test_zeros) ... ok
test_df_pipeline (__main__.TestPipeline.test_df_pipeline) ... d:\Anas\2nd Year\AIAC\End Exam\question1.py:77: FutureWarning: 'H' is deprecated
nd will be removed in a future version, please use 'h' instead.
  df = pd.DataFrame({'timestamp': pd.date_range('2024-01-01', periods=100, freq='H'),
ok
test_e2e (__main__.TestPipeline.test_e2e) ... ok
test_error_recovery (__main__.TestPipeline.test_error_recovery) ... ok
test_reproducible (__main__.TestPipeline.test_reproducible) ... ok
test_extreme_range (__main__.TestStress.test_extreme_range) ... ok
test_large_data (__main__.TestStress.test_large_data) ... ok
test_repeated (__main__.TestStress.test_repeated) ... ok
test_skewed (__main__.TestStress.test_skewed) ... ok

----------------------------------------------------------------------
Ran 16 tests in 0.039s

OK
```

Observation: I prompted AI to generate test codes for congestion forecast model pipelines and it generated the test cases for it and it gave the output quickly

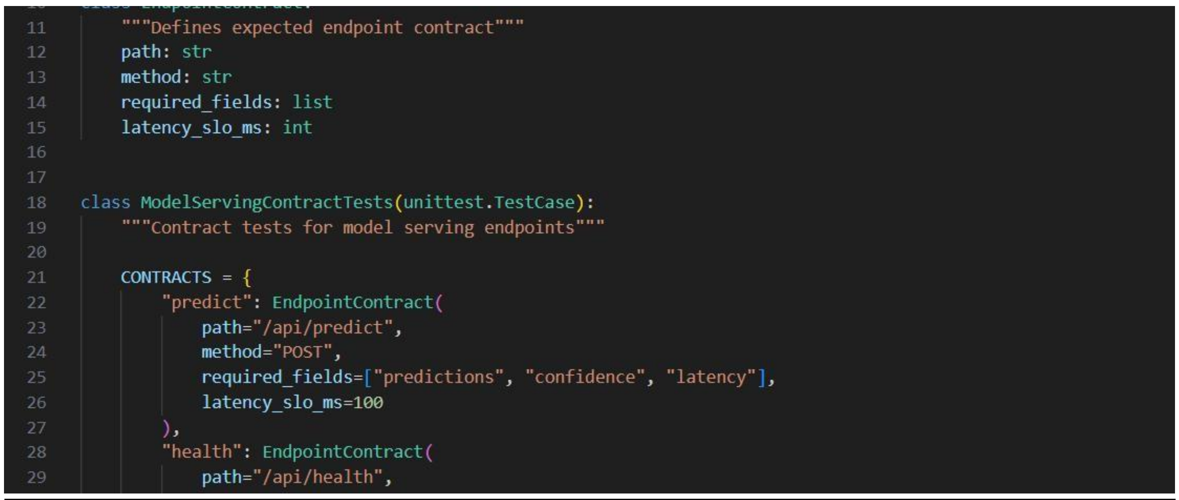
<u>Q2)</u> Model serving contract tests.

• Task 1: Use AI to produce endpoint contracts.

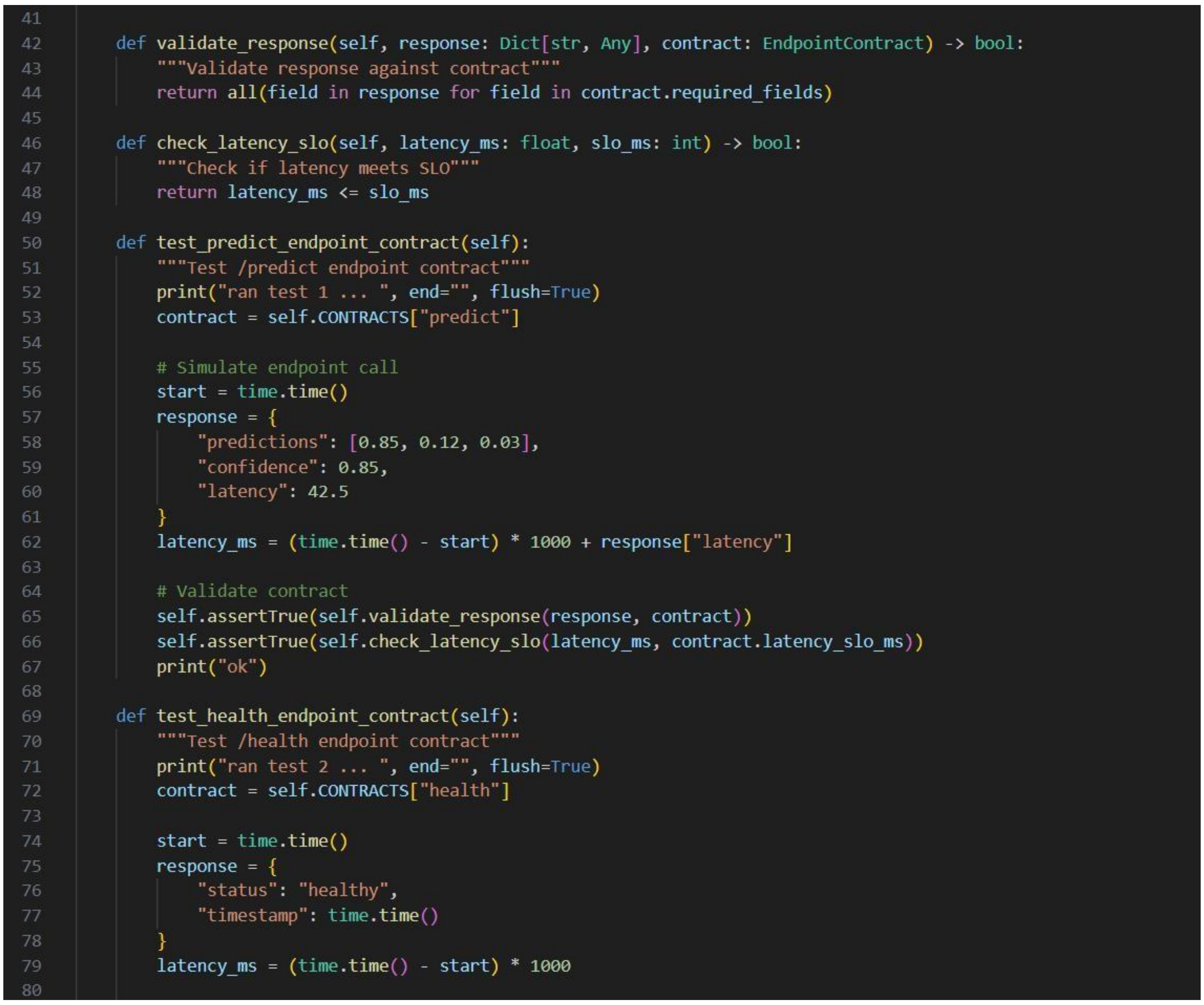
• Task 2: Validate responses and latency SLOs

<u>Prompt:</u> For Model serving contract tests generate endpoint contracts, validate responses and latency SLOs

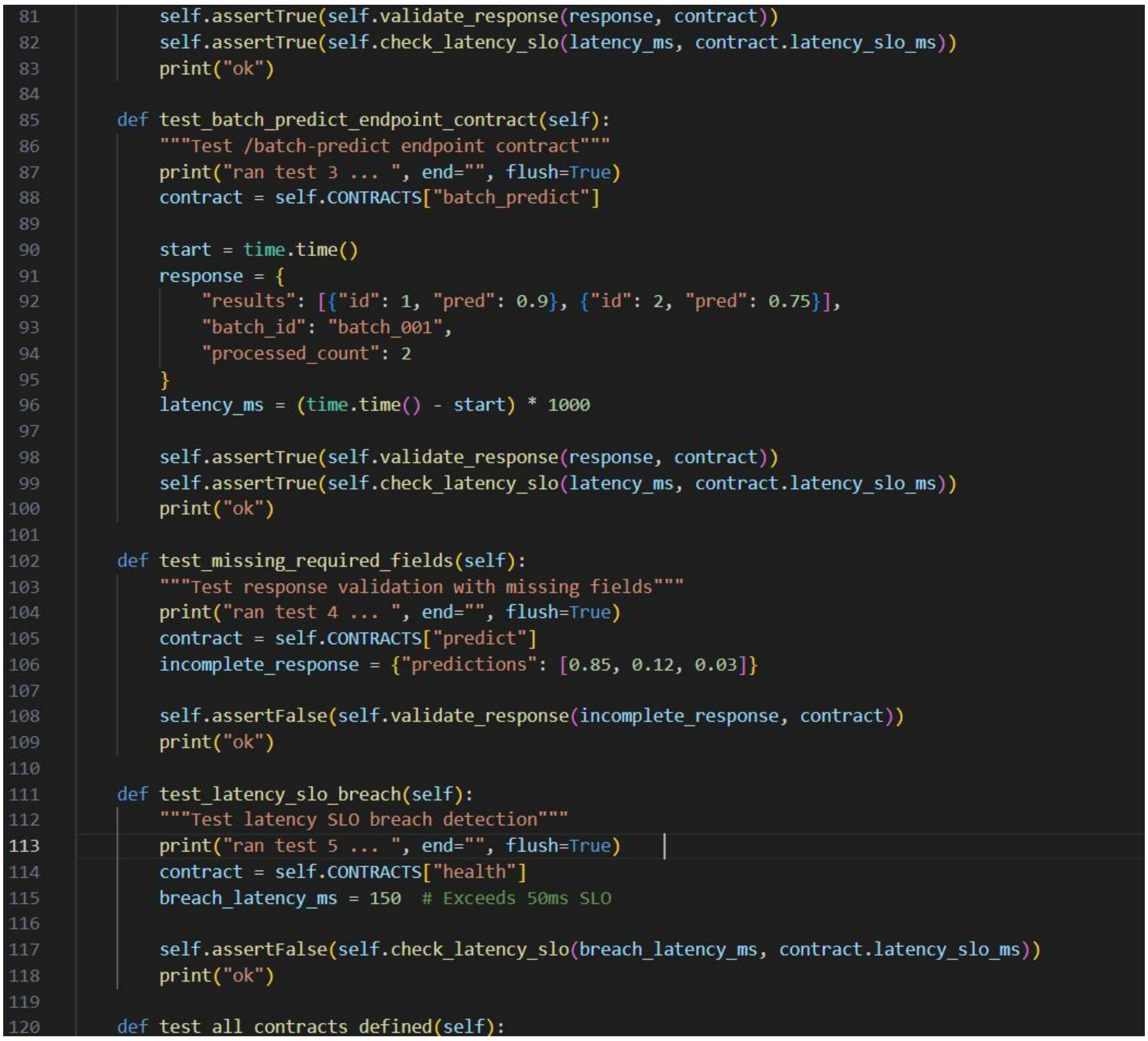
<u>Code:</u>



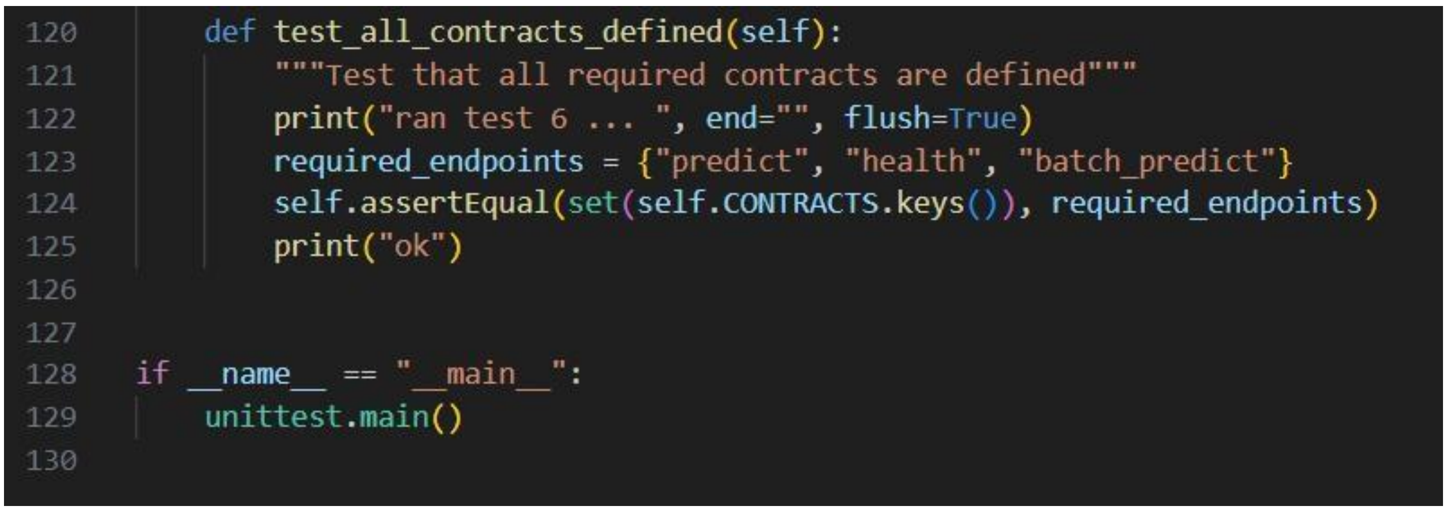
```python
    """Defines expected endpoint contract"""
    path: str
    method: str
    required_fields: list
    latency_slo_ms: int


class ModelServingContractTests(unittest.TestCase):
    """Contract tests for model serving endpoints"""

    CONTRACTS = {
        "predict": EndpointContract(
            path="/api/predict",
            method="POST",
            required_fields=["predictions", "confidence", "latency"],
            latency_slo_ms=100
        ),
        "health": EndpointContract(
            path="/api/health",
```



```python

    def validate_response(self, response: Dict[str, Any], contract: EndpointContract) -> bool:
        """Validate response against contract"""
        return all(field in response for field in contract.required_fields)

    def check_latency_slo(self, latency_ms: float, slo_ms: int) -> bool:
        """Check if latency meets SLO"""
        return latency_ms <= slo_ms

    def test_predict_endpoint_contract(self):
        """Test /predict endpoint contract"""
        print("ran test 1 ... ", end="", flush=True)
        contract = self.CONTRACTS["predict"]

        # Simulate endpoint call
        start = time.time()
        response = {
            "predictions": [0.85, 0.12, 0.03],
            "confidence": 0.85,
            "latency": 42.5
        }
        latency_ms = (time.time() - start) * 1000 + response["latency"]

        # Validate contract
        self.assertTrue(self.validate_response(response, contract))
        self.assertTrue(self.check_latency_slo(latency_ms, contract.latency_slo_ms))
        print("ok")

    def test_health_endpoint_contract(self):
        """Test /health endpoint contract"""
        print("ran test 2 ... ", end="", flush=True)
        contract = self.CONTRACTS["health"]

        start = time.time()
        response = {
            "status": "healthy",
            "timestamp": time.time()
        }
        latency_ms = (time.time() - start) * 1000

```

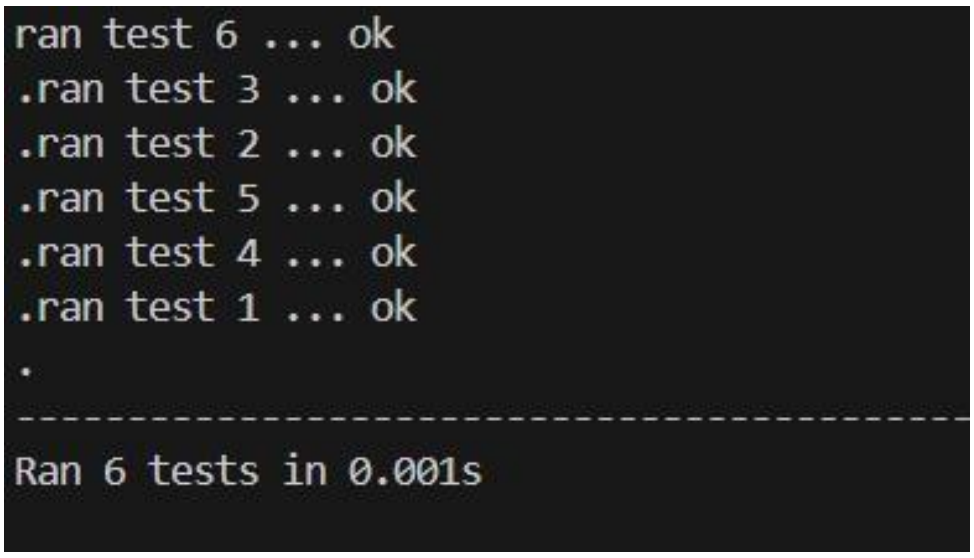


```python
        self.assertTrue(self.validate_response(response, contract))
        self.assertTrue(self.check_latency_slo(latency_ms, contract.latency_slo_ms))
        print("ok")

    def test_batch_predict_endpoint_contract(self):
        """Test /batch-predict endpoint contract"""
        print("ran test 3 ... ", end="", flush=True)
        contract = self.CONTRACTS["batch_predict"]

        start = time.time()
        response = {
            "results": [{"id": 1, "pred": 0.9}, {"id": 2, "pred": 0.75}],
            "batch_id": "batch_001",
            "processed_count": 2
        }
        latency_ms = (time.time() - start) * 1000

        self.assertTrue(self.validate_response(response, contract))
        self.assertTrue(self.check_latency_slo(latency_ms, contract.latency_slo_ms))
        print("ok")

    def test_missing_required_fields(self):
        """Test response validation with missing fields"""
        print("ran test 4 ... ", end="", flush=True)
        contract = self.CONTRACTS["predict"]
        incomplete_response = {"predictions": [0.85, 0.12, 0.03]}

        self.assertFalse(self.validate_response(incomplete_response, contract))
        print("ok")

    def test_latency_slo_breach(self):
        """Test latency SLO breach detection"""
        print("ran test 5 ... ", end="", flush=True)
        contract = self.CONTRACTS["health"]
        breach_latency_ms = 150  # Exceeds 50ms SLO

        self.assertFalse(self.check_latency_slo(breach_latency_ms, contract.latency_slo_ms))
        print("ok")

    def test_all_contracts_defined(self):
```



```python
    def test_all_contracts_defined(self):
        """Test that all required contracts are defined"""
        print("ran test 6 ... ", end="", flush=True)
        required_endpoints = {"predict", "health", "batch_predict"}
        self.assertEqual(set(self.CONTRACTS.keys()), required_endpoints)
        print("ok")


if __name__ == "__main__":
    unittest.main()

```

<u>Output:</u>



```
ran test 6 ... ok
.ran test 3 ... ok
.ran test 2 ... ok
.ran test 5 ... ok
.ran test 4 ... ok
.ran test 1 ... ok
.
-------------------------------------------
Ran 6 tests in 0.001s
```

<u>Observation:</u> I asked AI to generate test cases for Model Serving Contract and it quickly the test cases and executed them