

```
"""
```

```
🔪 SMOKE TEST – Corporate Emoji Translator  
72-hour feedback window harness.
```

```
Run:  python smoke_test.py  
Output: smoke_test_results.json (append-safe, one  
record per run)
```

```
"""
```

```
import json  
import time  
import random  
import hashlib  
from datetime import datetime, timezone  
from april_fools_emoji_swap import  
CorporateEmojiTranslator, LEXICON
```

```
# — Config
```

```
SMOKE_WINDOW_HOURS = 72  
RESULTS_FILE       = "smoke_test_results.json"  
RUN_ID             =  
hashlib.sha256(str(time.time()).encode()).hexdigest  
():8]
```

```
# — Test corpus
```

```
SENTENCES = [  
    "We need to deploy to production before the  
    deadline.",  
    "Let's circle back after the meeting to touch  
    base on the roadmap.",
```

```

    "The engineer found a bug in the legacy code
    during the sprint.",
    "Our agile process needs more synergy and
    stakeholder alignment.",
    "Going forward, let's leverage our ecosystem
    for scalable deliverables.",
    "The AI model caused an outage in production.
    Per my last email, monitoring is key.",
    "Anthropic and OpenAI are both disruptive and
    innovative.",
    "Action items: refactor the microservice
    pipeline before the next sprint.",
    "The kubernetes rollback failed. We need a deep
    dive with all stakeholders.",
    "Per my last email, the technical debt in our
    serverless architecture is not scalable.",
    # Edge cases
    "", # empty string
    "Hello world.", # no matches
    "DEPLOY TO PRODUCTION NOW!!!", # all-caps
    "deploy deploy deploy", # repeated word
    "Let's take this offline and circle back on the
    deep dive deliverable.", # multi-match
]

# — Helpers

```

```

def run_round_trip(translator, text):
    """Encode then decode; return pass/fail and
    details."""
    encoded = translator.encode(text)
    decoded = translator.decode(encoded)
    passed = decoded.lower() == text.lower()
    return {
        "original": text,
        "encoded": encoded,
        "decoded": decoded,
        "round_trip_pass": passed,
    }

```

```

    }

def check_coverage(translator):
    """Verify every LEXICON key produces at least
    one emoji substitution."""
    results = {}
    for phrase in LEXICON:
        encoded = translator.encode(phrase)
        substituted = encoded != phrase
        results[phrase] = substituted
    total = len(results)
    covered = sum(results.values())
    return {"total": total, "covered": covered,
    "uncovered": [k for k, v in results.items() if not
    v]}

def check_no_invisible_chars(text):
    """Fail if zero-width or other invisible
    Unicode is present."""
    invisible = ["\u200b", "\u200c", "\u200d",
    "\ufe0f", "\u00ad"]
    found = [ch for ch in invisible if ch in text]
    return {"pass": len(found) == 0, "found":
    found}

def run_fuzz(translator, n=50):
    """Random word soup – make sure it doesn't
    crash."""
    words = list(LEXICON.keys()) + ["random",
    "words", "that", "are", "not", "in", "lexicon"]
    errors = []
    for _ in range(n):
        sample = " ".join(random.choices(words,
        k=random.randint(1, 10)))
        try:
            translator.encode(sample)
            translator.decode(sample)
        except Exception as e:
            errors.append({"input": sample,
            "error": str(e)})

```

```

        return {"fuzz_rounds": n, "errors": errors,
                "pass": len(errors) == 0}

# — Main smoke test


---



def run_smoke_test():
    t_det =
CorporateEmojiTranslator(deterministic=True) #
reproducible
    t_rand =
CorporateEmojiTranslator(deterministic=False) #
real randomness

    print(f"\n🚬 Smoke Test Run ID: {RUN_ID}")
    print(f"    Started:
{datetime.now(timezone.utc).isoformat()}\n")

    # 1. Round-trip tests (deterministic)
    print("— 1. Round-trip fidelity


---

")
    rt_results = [run_round_trip(t_det, s) for s in
SENTENCES]
    rt_passed = sum(r["round_trip_pass"] for r in
rt_results)
    rt_failed = [r for r in rt_results if not
r["round_trip_pass"]]
    print(f"    Passed:
{rt_passed}/{len(rt_results)}")
    for f in rt_failed:
        print(f"        ❌ FAIL: {f['original'][:60]}")
        print(f"        decoded as: {f['decoded']
[:60]}")

    # 2. Lexicon coverage
    print("\n— 2. Lexicon coverage


---

")
    cov = check_coverage(t_det)
    print(f"    Covered:

```

```

{cov['covered']}/{cov['total']})")
    if cov["uncovered"]:
        print(f" ⚠️ Uncovered:
{cov['uncovered']}")
    else:
        print(" ✅ All phrases substituted
correctly.")

# 3. Invisible character audit
print("\n— 3. Invisible character audit
_____")
invis_issues = []
for s in SENTENCES:
    encoded = t_rand.encode(s)
    check = check_no_invisible_chars(encoded)
    if not check["pass"]:
        invis_issues.append({"input": s,
"found": check["found"]})
    if invis_issues:
        print(f" ❌ Found invisible chars in
{len(invis_issues)} outputs!")
        for issue in invis_issues:
            print(f" {issue}")
    else:
        print(" ✅ No invisible characters
found.")

# 4. Fuzz test
print("\n— 4. Fuzz test (50 random inputs)
_____")
fuzz = run_fuzz(t_rand)
if fuzz["pass"]:
    print(f" ✅ All {fuzz['fuzz_rounds']}
fuzz rounds passed.")
else:
    print(f" ❌ {len(fuzz['errors'])} errors
found:")
    for e in fuzz["errors"]:
        print(f" {e}")

```

```

# 5. Sample output for human review
print("\n— 5. Sample output (eyeball this)
      ")
samples = [
    "We need to deploy to production before the
deadline.",
    "Anthropic and OpenAI are both disruptive
and innovative.",
    "Per my last email, the technical debt is
not scalable.",
]
for s in samples:
    print(f"    IN: {s}")
    print(f"    OUT: {t_rand.encode(s)}\n")

# — Summary

```

```

overall_pass = (
    rt_passed == len(rt_results) and
    cov["covered"] == cov["total"] and
    len(invis_issues) == 0 and
    fuzz["pass"]
)

summary = {
    "run_id":      RUN_ID,
    "timestamp":
datetime.now(timezone.utc).isoformat(),
    "window_hours": SMOKE_WINDOW_HOURS,
    "overall_pass": overall_pass,
    "round_trip":   {"passed": rt_passed,
"total": len(rt_results), "failures": rt_failed},
    "coverage":     cov,
    "invisible_chars": {"pass":
len(invis_issues) == 0, "issues": invis_issues},
    "fuzz":         fuzz,
}

# Append to results file

```

```

try:
    with open(RESULTS_FILE, "r") as f:
        history = json.load(f)
    except (FileNotFoundError,
json.JSONDecodeError):
        history = []

    history.append(summary)
    with open(RESULTS_FILE, "w") as f:
        json.dump(history, f, indent=2,
ensure_ascii=False)

    print(f"\n— RESULT
_____")
    status = "✅ PASS" if overall_pass else "❌
FAIL"
    print(f" {status} (logged to
{RESULTS_FILE})\n")
    return overall_pass

if __name__ == "__main__":
    run_smoke_test()

```