1) Top section of the README (first 20–30 lines)

````md
# BreakPoint Library

Prevent bad AI releases before they hit production.

You change a model.
The output looks fine.
But:
- Cost jumps +38%.
- A phone number slips into the response.
- The format breaks your downstream parser.

BreakPoint catches it before you deploy.

It runs locally.
Policy evaluation is deterministic from your saved artifacts.
It gives you one clear answer:

`ALLOW` · `WARN` · `BLOCK`

## Quick Example

```bash
breakpoint evaluate baseline.json candidate.json
```

```text
STATUS: BLOCK

Reasons:
- Cost increased by 38% (baseline: 1,000 tokens -> candidate: 1,380)
```
````

---

2) Current API signature for `evaluate()`

```python
def evaluate(
    baseline_output: str | None = None,
    candidate_output: str | None = None,
    metadata: dict | None = None,
    baseline: dict | None = None,
    candidate: dict | None = None,
    strict: bool = False,
    config_path: str | None = None,
    config_environment: str | None = None,
    preset: str | None = None,
) -> Decision:
```

Return structure (`Decision`):
- `schema_version`
- `status` (`ALLOW|WARN|BLOCK`)
- `reasons`
- `reason_codes`
- `metrics`
- `metadata`
- `details`

---

3) Policy aggregator logic

```python
def aggregate_policy_results(results: list[PolicyResult], strict: bool = False)
-> Decision:
    reasons = []
    codes = []
    details = {}

    has_block = False
    has_warn = False
    for result in results:
        reasons.extend(result.reasons)
        codes.extend(result.codes)
        if result.status == "BLOCK":
            has_block = True
        elif result.status == "WARN":
            has_warn = True
        details[result.policy] = result.details or {}

    if has_block:
        status = "BLOCK"
    elif has_warn:
        status = "WARN"
    else:
        status = "ALLOW"

    if strict and status == "WARN":
        status = "BLOCK"
        reasons.append("Strict mode promoted WARN to BLOCK.")
        codes.append("STRICT_PROMOTED_WARN")

    reason_codes = [_to_reason_code(code) for code in codes]
    metrics = _extract_metrics(details)
    return Decision(status=status, reasons=reasons, reason_codes=reason_codes, m
etrics=metrics, details=details)
```

---

4) Implemented policies (cost, PII, drift, etc.)

Implemented policy evaluators:
- `evaluate_cost_policy(...)`
- `evaluate_pii_policy(...)`
- `evaluate_drift_policy(...)`
- `evaluate_output_contract_policy(...)`
- `evaluate_latency_policy(...)`

Locations:
- `breakpoint/engine/policies/cost.py`
- `breakpoint/engine/policies/pii.py`
- `breakpoint/engine/policies/drift.py`
- `breakpoint/engine/policies/output_contract.py`
- `breakpoint/engine/policies/latency.py`

---

5) CLI sample output (baseline/candidate)

Command:
```bash
python -m breakpoint.cli.main evaluate examples/quickstart/baseline.json example
s/quickstart/candidate_block.json
```

Output:
```text
══════════════════════════════════════
BreakPoint Evaluation
══════════════════════════════════════

Final Decision: BLOCK

Policy Results:
✗ No PII detected: Detected 1 match(es).
✓ Response format: No schema drift detected.
✗ Cost: Delta +40.00%.
✗ Latency: Delta +70.00%.
⚠ Output drift: Similarity 0.052632.

Summary:
- Cost increased by 40.0% (>35%).
- Latency increased by 70.0% (>60%).
- PII detected: EMAIL(1). Total matches: 1.
1 additional non-blocking signal(s) detected.

Exit Code: 0
══════════════════════════════════════
```

---

6) Default policy config (`default_policies.json`)

```json
{
  "cost_policy": {
    "min_baseline_cost_usd": 0.01,
    "warn_increase_pct": 15,
    "block_increase_pct": 35,
    "warn_delta_usd": 0.0,
    "block_delta_usd": 0.0
  },
  "pii_policy": {
    "patterns": {
      "email": "\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}\\b",
      "phone": "\\b(?:\\+?1[-.\\s]?)?(?:\\(?\\d{3}\\)?[-.\\s]?)\\d{3}[-.\\s]?\\d
{4}\\b",
      "credit_card": "\\b(?:\\d[ -]*?){13,16}\\b",
      "ssn": "\\b\\d{3}-\\d{2}-\\d{4}\\b"
    },
    "allowlist": []
  },
  "output_contract_policy": {
    "enabled": true,
    "block_on_invalid_json": true,
    "warn_on_missing_keys": true,
    "warn_on_type_mismatch": true
  },
  "drift_policy": {
```

```
    "warn_length_delta_pct": 75,
    "warn_short_ratio": 0.30,
    "warn_min_similarity": 0.10,
    "semantic_check_enabled": true,
    "similarity_method": "max(token_jaccard,char_3gram_jaccard)"
  },
  "latency_policy": {
    "min_baseline_latency_ms": 50,
    "warn_increase_pct": 25,
    "block_increase_pct": 60,
    "warn_delta_ms": 0.0,
    "block_delta_ms": 0.0
  },
  "model_pricing": {
    "gpt-4.1-mini": {
      "input_per_1k": 0.0004,
      "output_per_1k": 0.0016
    },
    "gpt-4.1": {
      "input_per_1k": 0.002,
      "output_per_1k": 0.008
    }
  }
}
```

---

7) How CLI parses flags like `--fail-on` and `--config`

Parsed in `breakpoint/cli/main.py`:
- `--config`: custom config path
- `--fail-on`: `choices=["warn","block"]`
- `--strict`, `--preset`, `--env`, `--json`, `--exit-codes`, `--now`

Behavior:
- `--config` is passed to `evaluate(... config_path=args.config ...)`
- `--fail-on warn` fails on WARN/BLOCK
- `--fail-on block` fails only on BLOCK

---

8) Test cases included in repo

Test files:
- `tests/test_evaluate.py`
- `tests/test_cli.py`
- `tests/test_cli_metrics.py`
- `tests/test_metrics.py`
- `tests/test_waivers.py`
- `tests/test_quickstart_samples.py`
- `tests/test_install_worthy_examples.py`
- `tests/test_ci_templates.py`
- `tests/test_packaging.py`
- `tests/test_baseline_lifecycle.py`

What they cover:
- Policy behavior (cost, pii, drift, latency, output_contract)
- Strict mode promotion
- Config/env overrides and validation
- CLI JSON/text/golden outputs and exit codes
- Metrics summarization

- Waivers
- Example reproducibility and packaging checks

---

9) `examples/` directory structure

```text
examples/ci/github-actions-breakpoint.yml
examples/ci/run-breakpoint-gate.sh

examples/install_worthy/baseline.json
examples/install_worthy/candidate_cost_model_swap.json
examples/install_worthy/candidate_format_regression.json
examples/install_worthy/candidate_killer_tradeoff.json
examples/install_worthy/candidate_pii_verbosity.json

examples/presets/chatbot.json
examples/presets/extraction.json
examples/presets/support.json

examples/quickstart/baseline.json
examples/quickstart/candidate_allow.json
examples/quickstart/candidate_block.json
examples/quickstart/candidate_warn.json
examples/quickstart/custom_policy.json
```

---

10) Errors / edge-case checks in current code

Main validation/error points:
- Missing required outputs:
  - `Baseline output is required.`
  - `Candidate output is required.`
- Waiver time required when waivers exist:
  - requires `metadata.evaluation_time` (ISO-8601)
- Combined CLI input validation:
  - must be object
  - must contain `baseline` and `candidate` objects
- Config validation:
  - unknown preset
  - invalid/missing environment overrides
  - non-numeric or inconsistent thresholds
  - invalid drift ranges
  - non-boolean output contract flags
  - invalid waivers schema/types
- Metrics validation:
  - requires at least one path
  - validates decision JSON schema keys and types
- CLI catches exceptions and emits:
  - JSON error payload with reason code, or
  - text `ERROR: ...`