Product Requirements Document: Prediction vs Reality Logger

1. Overview

The Prediction vs Reality Logger is a fully automated system to:

- Ingest forecast data from JSON files or external APIs
- Fetch actual market/event outcomes via pluggable data connectors
- Evaluate forecast accuracy (hit/miss) per scenario
- Store results in a time-series CSV for performance tracking
- Perform tensor-based analytics and generate human-readable summaries via ChatGPT
- Alert on any errors via Slack (with retries and secondary fallback)
- Deploy seamlessly using CLI, PowerShell script, Docker, and Kubernetes CronJobs
- Ensure code quality and reliability through unit tests

2. Goals & Success Metrics

Goal	Success Metric
Accurate ingestion & evaluation	100% of valid forecasts processed without errors
Extensible data connectors	New connector implemented in <2 hours
Tensor & LLM integration	Automated summary generated per day
Robust notifications	<1% missed alerts on failure
Deployment automation	Fully reproducible deployment via `start-app.ps1`, Docker, and k8s

3. Functional Requirements

1. **Forecast Loading**: Load `YYYY-MM-DD.json` files from `forecasts/` with validation
2. **Actuals Fetching**: Pluggable `ActualsSource` interface, default stub + API integration
3. **Accuracy Evaluation**: Support scenarios (`breakout`, `fade`, others) with clear rules
4. **Persistence**: Append daily results to `results.csv` using Pandas for resilience
5. **Tensor Analytics**: Load PyTorch model, run `predict`, integrate into workflow
6. **LLM Translation**: Convert tensor outputs into natural language summaries via ChatGPT
7. **Notifications**: Send alerts on errors with retries and fallback channels
8. **Configuration**: Central `config.yaml` supporting env overrides and validation
9. **CLI**: Single CLI with flags for date, dry-run, API fetch, tensor translation, verbosity
10. **Automation**: `start-app.ps1` for venv checks, and Kubernetes CronJob manifest for scheduling
11. **Packaging**: `requirements.txt`, `pyproject.toml`, Dockerfile, `venv/` structure

12. **Testing**: Unit tests covering all modules

4. Non-Functional Requirements

- **Reliability**: 99.9% uptime (CronJobs & Docker)
- **Maintainability**: 90% code coverage in tests
- **Performance**: Forecast processing in <1s, tensor/LLM in <5s
- **Security**: API keys via env variables, no secrets in code
- **Scalability**: Option to extend to multiple symbols or larger models

5. Architecture & File Structure

```
.
├── venv/                       # Python virtual environment
├── start-app.ps1               # Unified startup script
├── requirements.txt            # Pinned dependencies
├── config.yaml                 # Settings with env overrides
├── pyproject.toml              # Packaging & versioning
├── prediction_logger/          # Core package
│   ├── __init__.py             # Expose __version__
│   ├── config.py               # Loader & validation
│   ├── version.py              # Package version
│   ├── sources.py              # ForecastSource + JSON implementation
│   ├── logger.py               # Core run logic with error handling
│   ├── cli.py                  # Click-based CLI
│   ├── notifications.py        # Slack & secondary alerts
│   ├── thinkorswim.py          # Future socket client stub
│   ├── api_client.py           # External API integration
│   ├── tensor_model.py         # PyTorch model loader & predictor
│   └── translator.py           # LLM-based summaries
├── Dockerfile                  # Docker image build
├── k8s/                        # Kubernetes manifests
│   └── cronjob.yaml            # Daily schedule
├── tests/                      # Unit tests per module
├── dashboard/                  # Notebook stub for metrics
└── README.md                   # Usage guide
```

6. Seed Code for Top-Level Files

requirements.txt

```
click==8.1.3
PyYAML==6.0
pandas==2.0.3
```

```
python-dateutil==2.8.2
requests==2.31.0
websockets==11.0.3
torch==2.0.1
openai==0.27.0
```

config.yaml

```
timezone: "${TIMEZONE:-America/New_York}"
forecast_folder: "${FORECAST_FOLDER:-forecasts}"
output_csv: "${OUTPUT_CSV:-results.csv}"
schedule_time: "${SCHEDULE_TIME:-16:30}"
slack_webhook_url: "${SLACK_WEBHOOK_URL:-...}"
secondary_webhook_url: "${SECONDARY_WEBHOOK_URL:-}"
thinkorswim:
  host: "${TOS_HOST:-127.0.0.1}"
  port: ${TOS_PORT:-8200}
  use_ssl: ${TOS_USE_SSL:-false}
api:
  endpoint: "${API_ENDPOINT:-https://api.example.com/data}"
tensor:
  model_path: "${TENSOR_MODEL_PATH:-models/tensor_model.pt}"
```

start-app.ps1

```
<# Initialize and start application with checks #>
param([string]$ScriptDir = Split-Path -Parent
$MyInvocation.MyCommand.Definition)
Set-Location $ScriptDir
$venvActivate = "venv\Scripts\Activate.ps1"
if (-Not (Test-Path $venvActivate)) { Write-Host "Error: virtual env missing.";
exit 1 }
. $venvActivate
if (-Not (Test-Path "venv\Lib\site-packages\prediction_logger")) { pip
install . }
prediction-logger --verbose
```

pyproject.toml

```
[project]
name = "prediction_vs_reality_logger"
version = "0.1.0"
```

```
[tool.setuptools.package-data]
"prediction_logger" = ["config.yaml"]
```

7. Next Steps

- Review PRD with stakeholders
- Stub out missing data connectors
- Provide tensor model artifact
- Integrate CI/CD for tests and deployment
- Plan dynamic dashboards and multi-symbol support

8. Attachments: Full Module Code

prediction_logger/config.py

```
import os
import yaml

# Required configuration keys with defaults\...
```