برنامه ۴,۱۱: سند MLOps و مانیتورینگ — ShahnamehMap

نسخه: ۱,۰**

تاریخ: ۱۵/۰۹/۱۴۰۳**

تهیه‌کننده: مدیر فنی / مهندس ML**

وضعیت: فعال**

---

۱. فلسفه MLOps: یکپارچه‌سازی ایمن و مقیاس‌پذیر

هدف MLOps در **ShahnamehMap**، اتوماسیون، کنترل و نظارت بر چرخه حیات مدل‌های یادگیری ماشین** است تا اطمینان حاصل شود که مدل‌ها **به طور مداوم، قابل اعتماد و با کیفیت** در سرویس Production (نسخه‌بندی،) عمل می‌کنند. ما از **اصول مهندسی نرم‌افزار (CI/CD** برای سیستم‌های ML استفاده می‌کنیم، تست.

---

۲. چرخه حیات مدل (ML Lifecycle) و مراحل کلیدی

```mermaid
graph LR
A[نیاز کسب‌وکار / مشکل] --> B[آزمایش & توسعه];
B -- مدل + داده + کد --> C[استقرار & انتشار];
C --> D[مانیتورینگ & سرویس‌دهی];
D -- Drift/افت کیفیت --> E[اعتبارسنجی & بازآموزی];
E --> C;
D --> F[آرشیو & بازنشانی];
```

۳. زیرساخت و ابزارها (MLOps Stack)

لایه	**ابزار (انتخاب فعلی)**	**هدف**	**دلیل انتخاب**

| **ورژن‌کنترل کد & داده** | **GitLab** (کد)، **DVC (Data Version Control)** | ردیابی نسخه‌های مختلف داده، کد و داده/مدل) | سبک و مناسب داده‌های کوچک- DVC .موجود CI/CD مدل. | یکپارچه با متوسط. |

| ذخیره‌سازی مدل‌ها** (Model Registry)** | **DVC Remote** | پوشه ساختار یافته در** + **Storage** S3/MinIO** | با متادیتا. ذخیره، وابسته‌سازی و بازیابی مدل‌های آموزش‌دیده. | سادگی. در صورت نیاز ارتقا **MLflow** به قابلیت‌های پیشرفته (مانند مدل‌های مرحله‌بندی) به خواهیم داد. |

| خط لوله** CI/CD** | **GitLab CI/CD** + کانتینر** | خودکارسازی آموزش، تست و استقرار مدل. | یکپارچگی با **Docker** موجود DevOps زیرساخت. |

| محیط اجرا** (Runtime)** | در **Docker کانتینر** | ایزوله کردن و مقیاس سرویس مدل. | **Kubernetes (K8s)** استفاده از زیرساخت موجود. |

| استقرار** (Serving)** | **API REST** (FastAPI) | در کانتینر. | ارائه پیش‌بینی‌های مدل به سرویس‌های دیگر. | سادگی، سازگاری با معماری میکروسرویس. |

، (متریک) **Prometheus** | **مانیتورینگ & لاگینگ** |
لاگ)، **الگوریتم‌های تشخیص) **Loki**، (داشبورد) **Grafana**
سفارشی). | نظارت بر عملکرد، سلامت و کیفیت مدل. | استفاده از) **Drift
| .موجود Observability استک

برای آزمایش) +) **JupyterHub** | **اورکستراسیون/آزمایش** |
| .(Batch/برای اجرای دسته‌ای) **استاندارد Python **اسکریپت‌های
| .کنترل و انعطاف | .Batch محیط توسعه و اجرای

---

۴. نقش‌ها و مسئولیت‌ها (RACI Matrix)

| **DevOps مهندس** | **ML فعالیت** | **مهندس** |
| **مدیر محصول** | **مهندس داده**
| --- | --- | --- | --- | --- |
| (مشاوره زیرساخت) C | (انجام) **R** | **آزمایش و توسعه مدل** |
| (مشاوره داده) C | (تصویب نیاز) A

| **R** | (تعریف مراحل) R | **مدل CI/CD ایجاد خط لوله** |
| | | | (پیاده‌سازی)

تهیه) **R** | C | I | **R** | **آموزش و اعتبارسنجی مدل** |
| (داده

اجرای) **R** | (تأیید کیفیت) A | **انتشار و استقرار مدل** |
| | | | (استقرار

| (تحلیل، تنظیم آستانه) **R** | **مانیتورینگ و هشدار مدل** |
| C | I | (تنظیم ابزار) **R**

A | C | (پیشنهاد) **R** | **Rollback/تصمیم بازآموزی** |
| | (تصویب کسب‌وکار)

---

۵. خط لوله CI/CD برای مدل (ML Pipeline)

هر بار که کد مدل (`/ml/`) یا اسکریپت‌های آموزش در شاخه `main` یا `develop` آپدیت می‌شود، خط لوله زیر به طور خودکار اجرا می‌شود:

```yaml
# .gitlab-ci.yml (بخش ML)

stages:
  - test-ml
  - train-model
  - evaluate-model
  - build-model-image
  - deploy-to-staging

train-and-evaluate:
  stage: train-model
  script:
    - python train.py --config configs/model_v1.yaml  # آموزش با DVC
    - dvc metrics show  # نمایش معیارهای ارزیابی
    - python tests/model_tests.py  # اجرای تست‌های واحد مدل
  artifacts:
    paths:
```

```
    - model.pkl  # خروجی آموزش
    - metrics.json
  only:
    - main
    - develop
    - /^release-.*$/

deploy-model-staging:
  stage: deploy-to-staging
  script:
    - docker build -t shahnameh-rec-model:$CI_COMMIT_SHA .
    - kubectl set image deployment/ml-recommender model=shahnameh-rec-model:$CI_COMMIT_SHA -n staging
  only:
    - develop  # اتوماتیک به استیجینگ
```

:(Quality Gates) دروازه‌های کیفیت

۱. **Gate 1 (Unit Tests) تست کد):** عبور از تست‌های واحد.مدل

۲. **Gate 2 (تست داده):** بررسی کیفیت داده ورودی آموزش عدم) وجود Missingness توزیع منطبق ،شدید).

۳. **Gate 3 (معیارهای ارزیابی):** مدل جدید باید از لحاظ حداقل به اندازه **۹۵٪** **Recall@10** و **Precision@5** عملکرد داشته باشد. در غیر این صورت، خط لوله (Production) مدل قبلی متوقف و هشدار داده می‌شود.

۴. **Gate 4 (A/B Test در Staging):** قبل از Production ، مدل با **۱۰٪** از ترافیک واقعی** به مدت **حداقل Staging جدید در محیط تست می‌شود. تنها در صورت بهبود یا عدم تغییر معنی‌دار A/B **۴۸ ساعت راه می‌یابد Production ، به(CTR مانند) معیارهای کسب‌وکار.

---

۶. مانیتورینگ جامع در Production

مانیتور می‌کنیم Production ما چهار نوع مشکل اصلی را در مدل:

۶,۱. مانیتورینگ عملکرد سخت‌افزاری/زیرساختی (Infrastructure Monitoring)

* **متریک‌ها:** CPU/Memory مصرفی کانتینر مدل، زمان پاسخگویی API (Latency)، نرخ خطای HTTP (۵xx).
* **ابزار:** Prometheus (از طریق Exporterهای K8s).
* **هشدار:** اگر Latency P95 > 300ms برای ۵ دقیقه یا نرخ خطا > ۱٪.

۶,۲. مانیتورینگ دقت مدل (Model Performance Monitoring)

* **چالش:** در سیستم‌های توصیه‌گر، برچسب حقیقت زمینی (Ground Truth) بلافاصله در دسترس نیست (کاربر ممکن است هفته بعد با یک توصیه تعامل کند).
* **راهکار (Proxy Metrics):**
  * ردیابی نرخ کلیک (CTR) بر روی توصیه‌ها: Real-time.
  * نرخ تبدیل (Conversion Rate): نرخ شروع بازی پس از کلیک روی توصیه.

* **مشارکت (Engagement):** میانگین زمان صرف‌شده در بخش توصیه‌ها.

* **هشدار:** اگر CTR روزانه نسبت به میانگین متحرک ۷ روزه **بیش از ۲۵٪ افت** کند.

۶.۳ مانیتورینگ انحراف داده (Data Drift Monitoring)**

انحراف در توزیع (Feature Drift) می‌تواند عملکرد **ورودی‌های مدل** را خراب کند.

* **ویژگی‌های تحت نظارت:** `user_avg_rating`, `campaign_difficulty_tag`, `interaction_type_weight`.

* **روش:** مقایسه توزیع **ویژگی‌های هر درخواست ورودی** با توزیع **داده‌های آموزشی** (مرجع) با استفاده از **Production** **آماره‌های** PSI (Population Stability Index) یا KL Divergence به صورت روزانه.

* **هشدار:** اگر PSI برای هر ویژگی کلیدی **> ۰,۲** شود (نشان‌دهنده تغییر معنی‌دار).

۶,۴. مانیتورینگ انحراف مفهومی (Concept Drift Monitoring)

انحراف در رابطه بین **ورودی و خروجی**. برای ما، یعنی آیا کاربران هنوز به ویژگی‌های قدیمی (مثلاً "نژاد پهلوان") همانند گذشته واکنش نشان می‌دهند؟

روش:** ذخیره **یک نمونه تصادفی ۱٪ از درخواست‌های ** * پیش‌بینی** به همراه **نتیجه تعامل کاربر (مثلاً کلیک/عدم کلیک)** در روی **(Logistic Regression مثلاً) آینده. هر هفته، یک **مدل ساده مدلی که روی داده AUC آن را با AUC این داده جدید آموزش می‌دهیم و قدیمی آموزش دیده مقایسه می‌کنیم.

هشدار:** اگر **افت** AUC > ۰,۰۵** باشد.

---

۷. سیاست بازآموزی (Retraining Policy) و مدیریت نسخه**

۷,۱. بازآموزی برنامه‌ریزی‌شده (Scheduled Retraining):**

فرکانس:** **هفتگی** برای مدل توصیه‌گر (با داده فعلی). این ** *
.فرکانس با رشد داده می‌تواند تغییر کند

.GitLab CI در (Cron Job) زمان‌بند **:(Trigger) ماشه** *

به طور خودکار (`train-model` مرحله) CI/CD روند:** خط لوله** *
معیارهای ارزیابی)** عبور کرد،) Gate 3** اجرا می‌شود. اگر مدل جدید از
A/B** مستقر شده و وارد مرحله **Staging** به طور خودکار در
.می‌شود **Test

Event-Triggered) بازآموزی مبتنی بر رویداد ۷,۲.** ###
Retraining):**

:ماشه‌ها *

۱. هشدار** Data/Concept Drift شدید:** (PSI > 0.3 یا افت
.AUC > 0.1)

۲. افت شدید** Proxy Metrics:** (CTR ۴۰٪ > افت).

۳. تغییر اساسی در محصول:** راه‌اندازی یک ویژگی بزرگ که **
.الگوهای رفتار کاربر را تغییر می‌دهد

را مطلع می‌کند. آنها به صورت دستی خط ML روند:** هشدار، تیم** *
.لوله بازآموزی را راه‌اندازی و فرآیند را تسریع می‌کنند

۷,۳. مدیریت نسخه‌ها (Versioning):

* **قالب:** `MODEL_NAME-vMAJOR.MINOR.PATCH`

* **MAJOR:** تغییر معماری یا شکست در سازگاری (باعث می‌شود سرویس‌های مصرف‌کننده نیاز به به‌روزرسانی داشته باشند).

* **MINOR:** اضافه شدن ویژگی‌ها یا بهبود عملکرد (سازگار عقب‌رو).

* **PATCH:** رفع باگ یا بازآموزی با داده جدید (سازگار عقب‌رو).

ذخیره‌سازی:** هر نسخه از مدل، همراه با **کد، داده و ** در رجیستری مدل (DVC از طریق) هایپراپارامترهای** دقیقاً تکرارپذیر ذخیره می‌شود.

---

۸. استراتژی Rollback و بازیابی (Rollback & Recovery)

اصل:** توانایی بازگشت سریع به آخرین نسخه سالم مدل در کمتر از ** ۱۰** دقیقه.**

* **مدل‌های Production به عنوان Deployment‌های** روش: مدل‌ها مدیریت می‌شوند. مکانیزم **K8s** rollback است ساده آن:

```bash
kubectl rollout undo deployment/ml-recommender
```

این دستور، Deployment فعلی را به نسخه قبلی (stable)** برمی‌گرداند.

* **ماشه‌های Rollback دستی:**

۱. هشدارهای بحرانی مانیتورینگ که نشان‌دهنده خرابی کامل مدل است.

۲. شناسایی باگ‌های جدی در A/B Test استیجینگ.

۳. درخواست مدیر محصول به دلیل رفتار غیرمنتظره مدل.

* **Fallback State:** در طول Rollback (یا خرابی سرویس مدل)، سیستم به **الگوریتم هیورستیک ساده (Heuristic Fallback)** (مثلاً توصیه محبوب‌ترین‌ها + جدیدترین‌ها) تغییر حالت می‌دهد تا تجربه کاربر کاملاً قطع نشود.

---

۹. جمع‌بندی: قابل اعتماد و قابل نگهداری؟

ما در مسیر قابل اعتماد و قابل ML با استناد به این برنامه، **سیستم نگهداری بودن قرار دارد**، زیرا:

ریسک/اقدام	معیار بلوغ **ShahnamehMap**	**وضعیت**	**آینده**
تکرارپذیری (Reproducibility)	✓	عالی: کد، داده و مدل با DVC ورژن می‌شوند.	-
اتوماسیون (Automation)	✓	خوب: CI/CD برای آموزش و استقرار پایه وجود دارد.	نیاز به اتوماسیون کامل تشخیص و بازآموزی.
مانیتورینگ (Monitoring)	⚠	در حال توسعه: مانیتورینگ Proxy Metrics و Drift برقرار است.	مانیتورینگ زیرساخت و یکپارچه‌سازی هشدارها Drift در حال پیاده‌سازی.

خوب:** تست واحد و ** ✓ | **(Testing) تست و اعتبارسنجی** |
وجود دارد. | افزودن تست‌های یکپارچگی CI دروازه کیفیت مبتنی بر معیار در
| .خودکار A/B Test و (Integration)

Rollback خوب:** استراتژی** ✓ | **Rollback مدیریت خطا و** |
تعریف شده است. | مستندسازی دقیق‌تر سناریوهای Fallback ساده و
Rollback. |

اگر تغییرات **.Drift بزرگ‌ترین ریسک فعلی: تشخیص به موقع**
را دیر تشخیص دهیم، مدل به (Concept Drift) تدریجی در رفتار کاربران
قوی (بخش Drift تدریج بی‌اثر می‌شود. **پیاده‌سازی سیستم مانیتورینگ
**.۶,۳ و ۶,۴) اولویت فوری ماست

نتیجه:** ما از **بمب ساعتی** (مدل بدون نظارت) فاصله گرفته‌ایم و **
یک **چرخه حیات کنترل‌شده و مبتنی بر داده** برای مدل‌های خود ایجاد
کرده‌ایم. این چارچوب به ما امکان می‌دهد با اطمینان، مدل‌های بیشتری را در
.آینده توسعه و مستقر کنیم