MLOps و CML: من الفلسفة إلى التطبيق العملي

الفكرة الأساسية بسيطة: **MLOps هو الهدف، و CML هو إحدى أفضل الطرق للوصول إليه.**

- **MLOps (عمليات تعلم الآلة):** هي الفلسفة أو الإطار العام الذي يهدف إلى جعل عملية بناء ونشر نماذج تعلم الآلة موثوقة وقابلة للتكرار. إنه يجيب على سؤال **"ماذا"** نريد أن نحقق.
- **CML (تعلم الآلة المستمر):** هو النهج العملي والأدوات التي تطبق مبادئ MLOps، بالاعتماد بشكل كبير على أدوات تطوير البرمجيات الموجودة بالفعل. إنه يجيب على سؤال **"كيف"** نحقق ذلك.

جدول المقارنة: المبدأ مقابل التطبيق

مبدأ MLOps (الهدف)	كيف يطبقه CML (الأداة/الطريقة)
إمكانية إعادة الإنتاج (Reproducibility)	استخدام **Git** لتتبع إصدارات الكود، مع أداة مثل **DVC** لتتبع إصدارات البيانات والنماذج. هذا يضمن أن أي commit في Git يمكن إعادة بناء بيئته ونتائجه بالكامل.
الأتمتة (Automation)	استخدام منصات **CI/CD** (مثل GitHub Actions أو GitLab CI) لتشغيل خط أنابيب تعلم الآلة (التدريب، الاختبار، التقييم) تلقائيًا عند كل تغيير في الكود أو البيانات.
التعاون والمراجعة (Collaboration & Review)	يقوم CML بنشر **تقارير تلقائية** (تحتوي على مقاييس الأداء والرسوم البيانية) كتعليقات مباشرة في طلبات السحب (Pull Requests). هذا يسمح للمراجعين برؤية تأثير التغييرات دون الحاجة لتشغيل أي كود.
التكامل المستمر (Continuous Integration)	كل تغيير جديد يتم اختباره وتدريبه تلقائيًا للتأكد من أنه لا يقلل من أداء النموذج. هذا هو جوهر فكرة "CI for ML".

مثال عملي: تحسين نموذج تحليل المشاعر

تخيل أن لدينا فريقًا يعمل على نموذج يصنف مراجعات العملاء (إيجابية أم سلبية). النموذج الحالي يعمل بدقة 92%. أحد علماء البيانات يعتقد أنه يمكنه تحسين الدقة عبر تعديل طريقة معالجة النصوص.

الطريقة القديمة (بدون CML)

1. عالم البيانات ينسخ المشروع على جهازه.
2. يجري تعديلاته على الكود ويقوم بتدريب النموذج محليًا.
3. يحصل على دقة 94% ويشعر بالحماس.
4. يرسل الكود الجديد وملف النتائج (لقطة شاشة أو ملف نصي) إلى مدير الفريق عبر البريد الإلكتروني أو Slack.
5. المدير الآن بحاجة لسحب الكود، وتهيئة بيئة العمل، والتأكد من حصوله على نفس البيانات، ثم تشغيل التجربة بنفسه للتحقق من النتيجة. **هذه العملية بطيئة جدًا وعرضة للأخطاء.**

الطريقة الجديدة والمنظمة (باستخدام CML)

يتبع الفريق سير العمل التالي، الذي يتم إعداده مرة واحدة:

الخطوة 1: عالم البيانات يجري تغييره

1. ينشئ فرعًا جديدًا في Git: git checkout -b improve-text-processing.
2. يعدل الكود المسؤول عن معالجة النصوص.
3. يقوم بعمل commit و push للفرع الجديد، ثم يفتح **طلب سحب (Pull Request)** لدمج تغييراته في الفرع الرئيسي.

الخطوة 2: سحر CML و CI/CD يبدأ تلقائيًا

بمجرد فتح طلب السحب، يتم تشغيل سير عمل (Workflow) معرف مسبقًا في **GitHub Actions**. هذا السير يقوم بالآتي:

1. **التشغيل:** يتم تشغيل خادم افتراضي بالمواصفات المطلوبة.
2. **جلب الكود والبيانات:** يقوم الخادم بسحب الكود من الفرع الجديد، ويستخدم أمر dvc pull لجلب مجموعة البيانات الصحيحة.
3. **التدريب والتقييم:** يتم تنفيذ سكربت التدريب train.py. يقوم هذا السكربت بتدريب نموذجين: النموذج القديم (من الفرع الرئيسي) والنموذج الجديد (مع التعديلات).
4. **إنشاء التقرير:** بعد انتهاء التدريب، يتم إنشاء تقرير يقارن بين أداء النموذجين.

الخطوة 3: المراجعة السهلة واتخاذ القرار

يقوم CML بنشر هذا التقرير كتعليق تلقائي في صفحة طلب السحب. يرى مدير الفريق التعليق التالي:

🤖 **تقرير CML الآلي**

تمت مقارنة أداء هذا الفرع (improve-text-processing) مع الفرع الرئيسي (main).

المقياس	النموذج الرئيسي	هذا التعديل	التغيير
Accuracy	0.92	**0.94**	font> color="green">+ <0.02</font
F1 Score	0.91	**0.93**	font> color="green">+ <0.02</font

مصفوفة الارتباك (Confusion Matrix) الجديدة:

الاستنتاج: التعديلات في هذا الفرع أدت إلى تحسين أداء النموذج بشكل واضح.

النتيجة النهائية:

الآن، يمكن لمدير الفريق أن يرى بوضوح أن التغيير كان إيجابيًا. كل ما عليه فعله هو مراجعة الكود نفسه ثم الضغط على زر "Merge". العملية برمتها أصبحت شفافة، موثوقة، وسريعة للغاية.

هذا المثال يوضح كيف حولت CML مبادئ MLOps المجردة (الأتمتة، الموثوقية، التعاون) إلى خطوات عملية ومؤتمتة بالكامل.