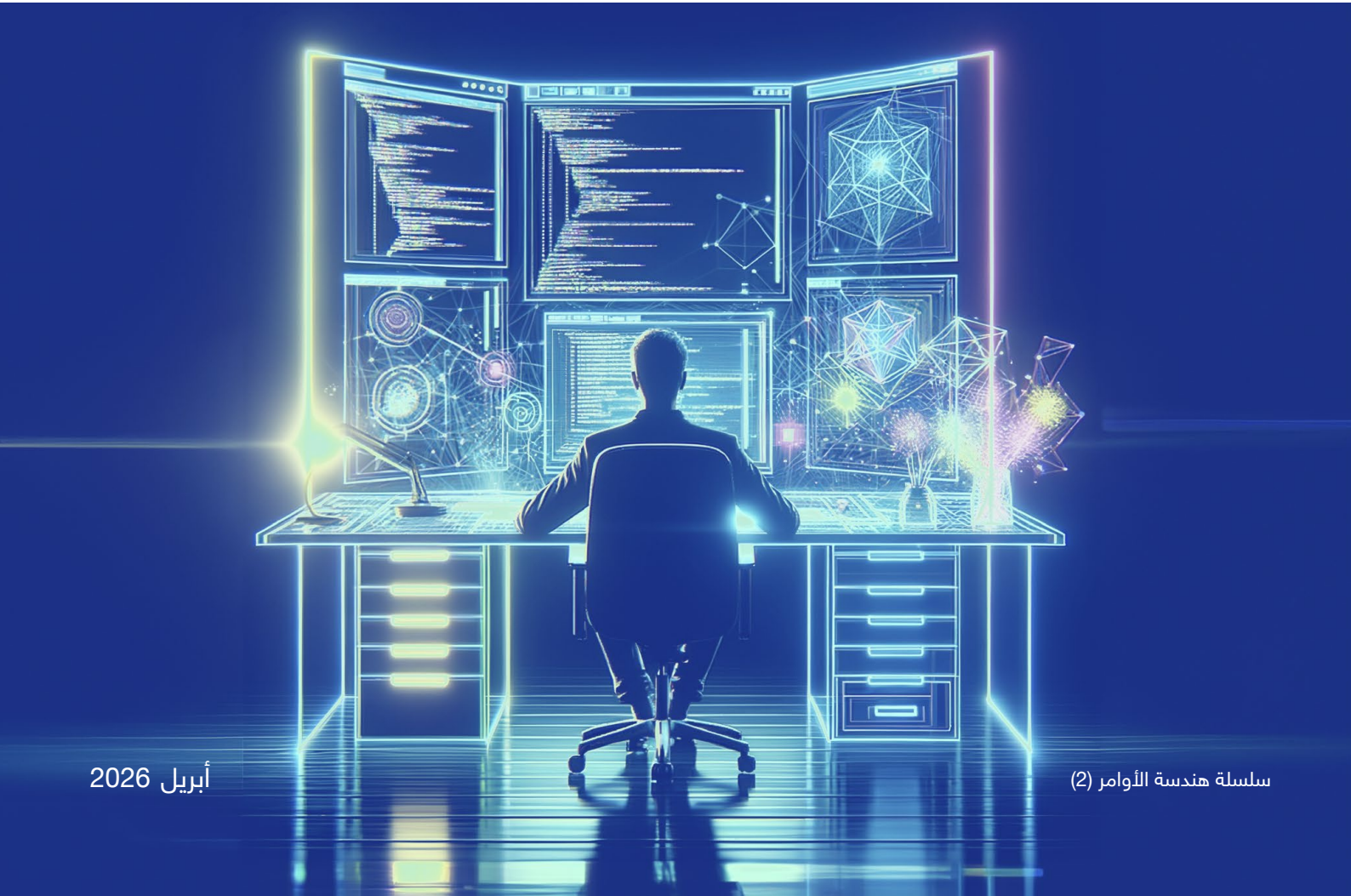
هندسة الأوامر

دليل المطورين



سلسلة هندسة الأوامر (2)

أبريل 2026

محتويات

مقدمة

أصبح الذكاء الاصطناعي مكوناً أساسياً في مراحل هندسة البرمجيات الحديثة، إذ لم يعد يقتصر دوره على أتمتة المهام البسيطة، بل أصبح يساهم في دعم التحليل، والتصميم، والاختبار، والصيانة. ومن خلال التكامل مع النماذج اللغوية الكبيرة القادرة على الفهم والتوليد والتعلّم من البيانات البرمجية، يشهد المجال تحولاً جذرياً أتاح للمطورين أدوات ذكية تعزز من الإنتاجية وتفتح آفاقاً جديدة للإبداع. كما يسهم هذا التطور في رفع جودة البرمجيات من خلال اكتشاف الأخطاء مبكراً واقتراح حلول فورية قائمة على المعرفة السابقة للنماذج. ويمكّن هذا التحول فرق التطوير من بناء أنظمة أكثر استقراراً وكفاءة بوقت أقل وجهد محدود، مع تحسين تجربة المستخدم النهائي بشكل مستمر.

يهدف هذا الدليل إلى تمكين مطوري البرمجيات من إتقان أسس ومهارات هندسة الأوامر، بما يمكّنهم من صياغة تعليمات دقيقة وفعّالة تُساهم في تحقيق أفضل المخرجات من النماذج اللغوية وتوظيفها بفعالية في تطوير البرمجيات وتحسين أداء التطبيقات. كما يستعرض الدليل تطبيقات عملية وأمثلة واقعية لهندسة الأوامر في سياقات تطوير البرمجيات، مثل توليد الأكواد البرمجية، ودعم عمليات ضمان الجودة والاختبار، وتعزيز أمن الأنظمة، وغيرها من المهام ذات الصلة. ويركز الدليل كذلك على الأبعاد الأخلاقية ومبادئ الاستخدام المسؤول، لضمان توظيف هذه التقنيات بشكل آمن وموثوق. وبذلك، يقدم الدليل إطاراً متكاملاً يجمع بين المعرفة النظرية والتطبيقات العملية لدعم المطورين في بناء حلول مبتكرة قائمة على الذكاء الاصطناعي.

النماذج اللغوية الكبيرة

تشهد النماذج اللغوية الكبيرة تطوراً نوعياً جعلها في صدارة إنجازات الذكاء الاصطناعي التوليدي، إذ لم يعد دورها مقتصراً على معالجة النصوص فحسب؛ بل توسع ليشمل أنماطاً متعددة من البيانات كالأصوات والصور ومقاطع الفيديو. تستند هذه النماذج إلى بنية المحوِّل (Transformer) التي أحدثت نقلة نوعية في تحليل تسلسل الكلمات والتنبؤ بالمدخلات التالية، معتمدة على مليارات المعاملات وكميات ضخمة من البيانات، مما يمكِّنها من فهم مكونات اللغة وترابطها.

لا تقتصر هذه النماذج على بنيتها، بل تمتد إلى قابليتها للتكيّف وفق مهام مختلفة عبر تقنيات متعددة، مثل الضبط الدقيق (Fine Tuning)، والتعلم داخل السياق (In-Context Learning)، والتعلم بعدد قليل من الأمثلة (Few-Shot Learning). وقد ازداد الاهتمام بهذه النماذج في السنوات الأخيرة باعتبارها أنظمة موحدة ومتعددة الأغراض يمكن توجيهها من خلال أوامر مكتوبة بلغة طبيعية لتنفيذ وظائف متنوعة. ومع ذلك، تعتمد جودة مخرجات النماذج اللغوية على دقة صياغة الأوامر (Prompts) الموجَّهة إليها، إذ إن الأوامر المصاغة بطريقة جيدة غالباً ما تُسهم في تحقيق نتائج أكثر اتساقاً وجودة. وقد أدى ذلك إلى ظهور هندسة الأوامر (Prompt Engineering) كإحدى المهارات المحورية في التفاعل مع هذه النماذج، ليس فقط للمهندسين والباحثين في مجالات الذكاء الاصطناعي، بل أيضاً للمتخصصين في قطاعات مهنية مختلفة مثل التعليم والرعاية الصحية، مما يعكس الدور المتنامي لهذه النماذج في مختلف المجالات.

هندسة الأوامر وأهميتها

تشير هندسة الأوامر (Prompt Engineering) إلى عملية تصميم وتحسين الأوامر بهدف توجيه النماذج اللغوية الكبيرة نحو توليد المخرجات المطلوبة بدقة وفعالية. فمن خلال صياغة الأوامر بجودة عالية، يتم تزويد النموذج بالسياق والتعليمات والأمثلة التي تساعده على فهم المقصود والاستجابة بالطريقة المُثلى التي يرغب بها المستخدم. كما تُمكن هندسة الأوامر المطورين والباحثين من تحسين أداء النماذج في مهام محددة، واستكشاف تطبيقات جديدة، وتوفير الوقت والموارد، إذ تُعد الأوامر بمثابة "برمجة" غير مباشرة للتحكم بمخرجات النماذج وتخصيصها. كما توفر هندسة الأوامر إطاراً لتوثيق أنماط خاصة ببنية الأوامر لمعالجة مهام متعددة تساعد على تكييف النماذج في مجالات مختلفة.

ومع التوسع السريع في استخدام النماذج اللغوية، يتزايد دور هندسة الأوامر كأداة أساسية لفهم إمكانات هذه النماذج وتوظيفها بفعالية في التطبيقات العملية، مثل المساعدات الافتراضية، وبوتات المحادثة، وأدوات الذكاء الاصطناعي الحواري. ومن المتوقع أن تسهم هندسة الأوامر في تطوير واجهات أكثر وضوحاً للتحكم بالمخرجات، وفتح فرص جديدة في سوق العمل، مما يجعلها عنصراً محورياً في إطلاق القدرات الكاملة للنماذج اللغوية الكبيرة.

مميزات هندسة الأوامر

تُعد هندسة الأوامر أداة أساسية لتعزيز قدرات النماذج اللغوية وتوسيع مجالات استخدامها، إذ تتيح عدداً من الفوائد، من أبرزها:

- **تحسين أداء النموذج:** يُسهم تصميم الأوامر بدقة في الحصول على مخرجات أكثر دقةً وثراءً بالمعلومات، من خلال تزويد النموذج بتعليمات واضحة وسياق محدد.
- **تقليل التحيز والمخرجات غير المرغوبة:** تساعد هندسة الأوامر على الحد من التحيز وتقليل احتمالية توليد محتوى مسيء أو غير مناسب.
- **تقليل الغموض:** تساعد الأوامر المصاغة بدقة على توضيح السياق وتقليل الغموض، مما يقلل من هلوسة الاستجابات ويعزز الموثوقية.
- **التحكم الفعَّال في المخرجات:** تتيح هندسة الأوامر توجيه سلوك النموذج نحو استجابات أكثر اتساقاً وقابلية للتنبؤ بما يعكس الهدف المطلوب.
- **تحسين تجربة المستخدم:** توفر الأوامر الواضحة والمباشرة تجربة أكثر سلاسة وسهولة في التفاعل مع النماذج، مما ينعكس إيجاباً على رضا المستخدم وجودة تفاعله.

أبرز حالات الاستخدام

تتيح هندسة الأوامر للمطورين الاستفادة من النماذج اللغوية المتقدمة في تحسين الإنتاجية، وضمان جودة الأكواد، وتسريع التعلم. فيما يلي أبرز حالات الاستخدام:

البرمجة

- توليد الأكواد الأولية وأتمتة الأكواد المتكررة لتقليل وقت التطوير.
- إعادة هيكلة الأكواد وتحسين بنيتها لتعزيز قابلية الصيانة.
- التكامل مع واجهات برمجة التطبيقات (API Integration) لتوليد أكواد الربط مع الخدمات الخارجية وتوثيق نقاط النهاية (End Points).

ضمان الجودة

- المساعدة على تصحيح الأخطاء عبر تحليل رسائل الخطأ أو السجلات (Logs) واقتراح الحلول.
- توليد الاختبارات (Test Generation) لتغطية سيناريوهات الاستخدام المختلفة.
- المساعدة على تتبع الأخطاء وتحليل أسبابها الجذرية.

تعزيز الأمان

- اكتشاف الثغرات في الأكواد البرمجية واقتراح حلول بديلة.
- اقتراح تحسينات لتعزيز الأمان في قاعدة البيانات.
- توليد اختبارات أمان من خلال إنشاء حالات اختبار لمحاكاة الهجمات والتحقق من أمان التطبيقات.

التعلم والتطوير

- تعلم لغات برمجة وأطر عمل جديدة.
- توثيق الأكواد وإضافة التعليقات التوضيحية.
- الهندسة العكسية للأكواد لفهم وتحليل المشاريع القديمة أو غير الموثقة.

أساسيات هندسة الأوامر

يُعد الإلمام بأساسيات هندسة الأوامر خطوة محورية لفهم كيفية التفاعل الفعّال مع النماذج اللغوية والتحكّم في سلوكها وجودة مخرجاتها. يستعرض هذا القسم المفاهيم الأساسية لهندسة الأوامر، بما يشمل أنواع الأوامر، وعناصر بناء الأمر، وطريقة بناء الأوامر، مما يوفّر أساساً عملياً لتطبيق هذه المفاهيم في سياقات تطوير البرمجيات.

أنواع الأوامر

في سياق هندسة الأوامر، يُستخدم مصطلحا أوامر المستخدم (User Prompts) وأوامر النظام (System Prompts) للإشارة إلى نوعين مختلفين من التوجيهات التي يتعامل معها النموذج اللغوي أثناء توليد الاستجابة.

تشير أوامر المستخدم إلى التعليمات المباشرة أو الأسئلة التي يُقدمها المستخدم أثناء التفاعل مع النموذج، وتُركز على تحقيق مهمة محددة أو إنتاج مخرجات معينة في لحظة التنفيذ. في المقابل، تشير أوامر النظام إلى مجموعة من التعليمات التي يضعها المطوّر للتحكم في سلوك النموذج وتحديد قواعده العامة، وتنفّذ في الخلفية دون أن تكون مرئية للمستخدم النهائي. ولا يمكن للمستخدم الاطلاع على أوامر النظام أو تعديلها أو تجاوزها من خلال أوامره الخاصة، إذ تُطبق لضمان اتساق السلوك، وأمان المخرجات، والالتزام بالسياسات المعتمدة.

وبشكل عام، تُحدد أوامر النظام كيف يتصرّف النموذج، بينما تُحدد أوامر المستخدم ما الذي يُطلب من النموذج فعله، ويُعد الجمع بينهما من أفضل الممارسات لضمان اتساق المخرجات وجودتها في بيئات التطوير والتفاعل الذكي. كما أن استخدام أوامر النظام يساعد على:

- توليد استجابات أكثر تخصيصاً ودقة.
- ضمان الالتزام بإرشادات محددة وثابتة خلال التفاعل بين المستخدم والنموذج.
- تشغيل أوامر متعددة بنفس مجموعة التعليمات الثابتة دون الحاجة إلى تكرارها في كل مرة.
- تزويد النموذج بمعلومات لا يمكن للمستخدم النهائي رؤيتها أو تعديلها.

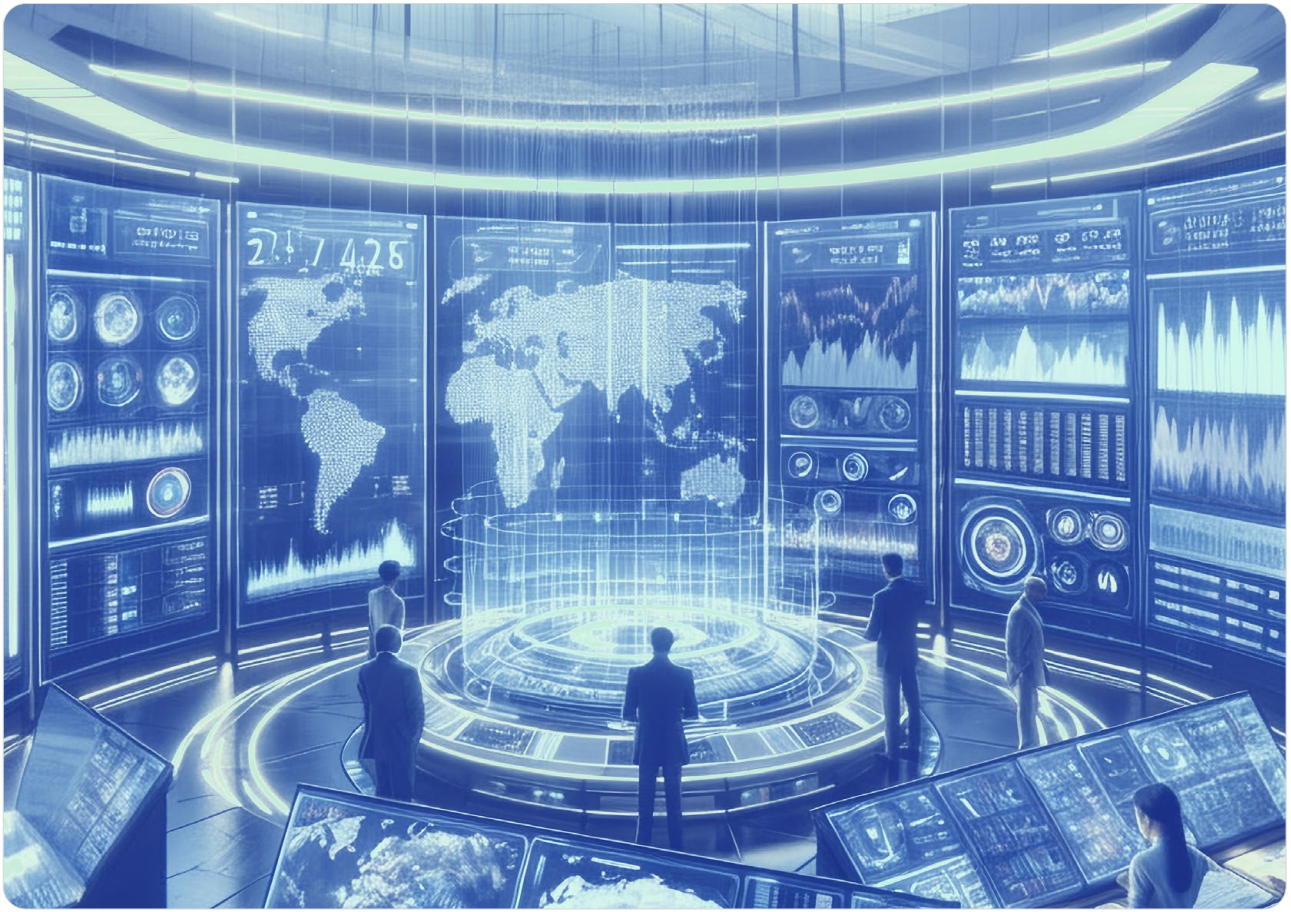
وتُطبق أوامر النظام أولاً لضبط سلوك النموذج وسياقه العام، ثم تُنفذ أوامر المستخدم بعد ذلك لتنفيذ المهام المحددة ضمن الإطار الذي حددته أوامر النظام. يختلف مكان تطبيق هذه الأوامر بحسب السياق التقني، ومن أبرز الطرق:

- الواجهات الرسومية (GUI) للنماذج اللغوية مثل (OpenAI Platform) و(Vertex AI Studio)، وذلك من خلال ضبط الإعدادات وتحديد سلوك النموذج في بداية المحادثة.
- في التطبيقات المدمجة (Integrated Systems) التي تعتمد على النماذج اللغوية، مثل المساعدات الذكية، تُدمج تعليمات النظام في الكود البرمجي، وغالباً ما يتم تطبيق ذلك عبر الواجهات البرمجية (API) للنماذج اللغوية مثل (OpenAI) و (Vertex AI) وتضمين الأوامر في حقول مخصصة داخل الطلب، مثل:

```
{
“role”: “system”,
“content”: “أنت مساعد ذكاء اصطناعي متخصص في توليد
الأكواد البرمجية”
}
{
“role”: “user”,
“content”: "أنشئ كود بايثون يقوم بعملية الضبط الدقيق لنموذج
BERT "
}
```

عناصر بناء الأوامر

تتأثر جودة الأمر وفعاليته بجانبين رئيسيين: المحتوى والهيكلة. يركز المحتوى على تزويد النموذج بجميع المعلومات المتعلقة بالمهمة، مثل التعليمات، والأمثلة، والمعلومات السياقية، وغيرها. بينما تشير الهيكلة إلى كيفية تنظيم المحتوى بطريقة واضحة، مثل طريقة عرض المعلومات واستخدام القوالب، مما يؤثر في جودة الاستجابة. وعلى الرغم من عدم وجود طريقة صحيحة أو خاطئة لتصميم الأوامر، إلا أن هناك استراتيجيات شائعة يمكن أن تحسّن من أداء النموذج في تقديم الاستجابة المطلوبة. يستعرض **الجدول (1)** أبرز هذه العناصر، حيث تنقسم إلى عناصر أساسية تشمل الهدف والتعليمات الرئيسية، وأخرى اختيارية ترتبط بالتحكم في سلوك النموذج وأسلوبه قبل توليد الاستجابة، التي غالباً ما يشار إليها بأوامر النظام.



الجدول (1): العناصر الأساسية والاختيارية لبناء الأوامر

العنصر	الوصف
العناصر الأساسية	
الهدف	المهمة التي يهدف المستخدم إلى تحقيقها، ويكون الهدف إما محدداً أو عاماً. مثال: توليد دالة بلغة (Python) تقوم بترتيب قائمة المستخدمين حسب تاريخ إنشاء الحساب.
التعليمات	تعليمات لكيفية أداء المهمة المستهدفة. مثال: اكتب الكود باستخدام لغة (Python)، مع شرح مختصر لكل خطوة، وراعِ أفضل ممارسات الأداء.
العناصر الاختيارية	
المعلومات السياقية	المعلومات التي يعتمد عليها النموذج لأداء المهمة. يطلق عليها أيضاً بـ"المستندات" أو "بيانات الإدخال". مثال: هذه الدالة ستُستخدم في نظام (backend) يعتمد على (Django) ويتعامل مع آلاف السجلات.
تعيين الدور	تعيين دور معين للنموذج اللغوي، مما يساهم في توليد إجابات ذوات شخصية واضحة وملائمة للدور المطلوب. مثال: تصرّف كمطور برمجيات خبير في (Python) وتحسين الأداء.
القيود	القيود أو الضوابط التي يجب على النموذج الالتزام بها عند توليد الاستجابة، التي تشمل ما يمكن وما لا يمكن للنموذج القيام به. مثال: لا تستخدم مكتبات خارجية، ويجب أن يكون الحل متوافقاً مع (Python 3.10).
النبرة	تحديد النبرة أو الأسلوب الذي ينبغي للنموذج محاكاته أثناء توليد الاستجابة. مثل أن يجيب بطريقة علمية، أو بأسلوب عامي. مثال: استخدم أسلوباً تقنياً احترافياً موجّهاً لمطوري البرمجيات.
تنسيق الاستجابة	صيغة الاستجابة التي يحددها المستخدم. مثل أن تكون الإجابة بصيغة (JSON)، أو جدول، أو فقرة، أو قائمة نقطية. يطلق عليها أيضاً بـ"الهيكل" أو "العرض". مثال: قدّم الإجابة على شكل كود برمجي، ثم قدّم شرحاً مختصراً بنقاط.
الملخص	إعادة موجزة لأهم التعليمات في نهاية الأمر، خصوصاً القيود وصيغة الاستجابة. باختصار: قدّم دالة (Python) محسّنة، دون استخدام لمكتبات خارجية، مع شرح تقني واضح.

طريقة بناء الأوامر

ينتشر استخدام مصطلحي الأمر (Prompt) وقالب الأمر (Prompt Template) في مجال هندسة الأوامر كمفهومين مرتبطين ببناء وكتابة الأمر للنماذج اللغوية، ويستعرض هذا القسم الفرق بينهما، ودواعي الاستخدام لكل منهما.

الأمر

تعليمات محددة تُرسل مباشرة للنموذج اللغوي. يمكن أن يحتوي الأمر على أسئلة، أو معلومات سياقية، أو أمثلة، أو نص جزئي يُطلب من النموذج إكماله.

مثال: "اكتب دالة (Python) للتحقق من صحة مدخلات نموذج تسجيل مستخدم."

خصائص الأمر:

- يُكتب للاستخدام مرة واحدة.
- خاص بالمهمة الحالية.
- قد يتضمن أسئلة أو معلومات سياقية أو صيغة المخرجات.
- لا يمكن إعادة استخدامه بسهولة دون إعادة الصياغة.

قالب الأمر

يشير قالب الأمر إلى هيكل نصي (Textual Structure) مُعد مسبقاً لتوجيه النموذج اللغوي نحو أداء مهمة محددة. يعد نسخة عامة من الأمر يتضمن متغيرات قابلة للتعديل (Placeholders)، ويمكن إعادة استخدامه بمجرد تغيير القيم. يعتمد قالب الأمر على مزيج من محتوى ديناميكي يتغير حسب السياق ومدخلات المستخدم، ومحتوى ثابت يُحدد التعليمات العامة وأسلوب الصياغة للأمر.

يُنصح باستخدام قالب الأمر عندما يُتوقع حصول تكرار في أي جزء من الأمر، مما يساهم في أتمتة المهام، كما يُستخدم عادة في التطبيقات المعتمدة على النماذج اللغوية الكبيرة، وذلك عبر ربطه بالواجهات البرمجية لهذه النماذج، مثل (OpenAI API) و(Claude API).

مثال:"راجع الكود البرمجي التالي بلغة {اللغة}، وحدد الأخطاء المحتملة، ثم اقترح تحسينات تتوافق مع أفضل الممارسات: {الكود}"

حيث تشير {اللغة} إلى اللغة البرمجية كعنصر متغير يتم استبداله باللغة المستهدفة. ويشير {الكود} إلى الكود البرمجي المطلوب مراجعته.

خصائص قالب الأمر:

- قابل لإعادة الاستخدام.
- يحتوي على عناصر متغيرة تعتمد على مدخلات المستخدم.
- يحتوي على محتوى ثابت لتوجيه النموذج إلى تحقيق أهداف الأمر.
- يستهدف المهام العامة والمتخصصة.

فوائد استخدام قوالب الأوامر:

الاتساق

ضمان هيكل ثابت للأوامر عبر تفاعلات متعددة.

الكفاءة

إمكانية استبدال المحتوى المتغير بسهولة دون الحاجة إلى إعادة كتابة الأمر.

سهولة الاختبار

اختبار مدخلات مختلفة وتقييم الاستجابات فورياً من خلال تغيير الجزء المتغير فقط.

قابلية التوسع

تساعد التطبيقات المعتمدة على النماذج اللغوية على تنظيم الأوامر بمرونة، مما يؤدي إلى سهولة إدارة التطبيق مع ازدياد نموه.

طرق استخدام قوالب الأمر:

يمكن استخدام قوالب الأمر عبر بيئات مختلفة، وذلك بحسب طبيعة المشروع أو الأداة المستخدمة، ومن أبرز الطرق لاستخدام قوالب الأمر:

- **الواجهات الرسومية (GUI)**

توفر كثير من منصات النماذج اللغوية واجهة محادثة رسومية يمكن إدخال القوالب فيها مباشرة والتحكم بالمتغيرات والأدوات المساعدة، مثل (OpenAI Platform) و(Anthropic Console).

- **الواجهات البرمجية (API)**

يعد هذا الأسلوب الأكثر استخداماً لدى المطورين، لمرونته وسهولة الدمج مع التطبيقات الخارجية. يستعرض المثال التالي كيفية الربط مع الواجهة البرمجية للنموذج (GPT-4) باستخدام لغة (Python)، وإرسال الأمر مع تعليمات النظام إلى النموذج.

```
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")
response = client.chat.completions.create(
  model="gpt-٤",
  messages=[
     {"role": "system", "content": "أنت مساعد ذكي يجيب بإيجاز وباللغة العربية."},
     {"role": "user", "content": "أشرح ما هو الذكاء الاصطناعي؟"}
 ]
)
print(response.choices[٠].message.content)
```

وفي المثال التالي، يستعرض الكود كيفية بناء صيغة عامة للقالب، بإضافة محتوى ثابت مثل أسلوب الرد، ومحتوى متغير يحمل مدخلات المستخدم أو بيانات من مصادر خارجية.

```
template = """
أنت خبير في التجارة الإلكترونية.
اكتب إجابة واضحة ومبسطة.
سؤال المستخدم: {question}
"""
user_question = "ما فوائد التجارة الإلكترونية للشركات الصغيرة؟"
prompt = template.format(question=user_question)
```

أدوات وسيطة

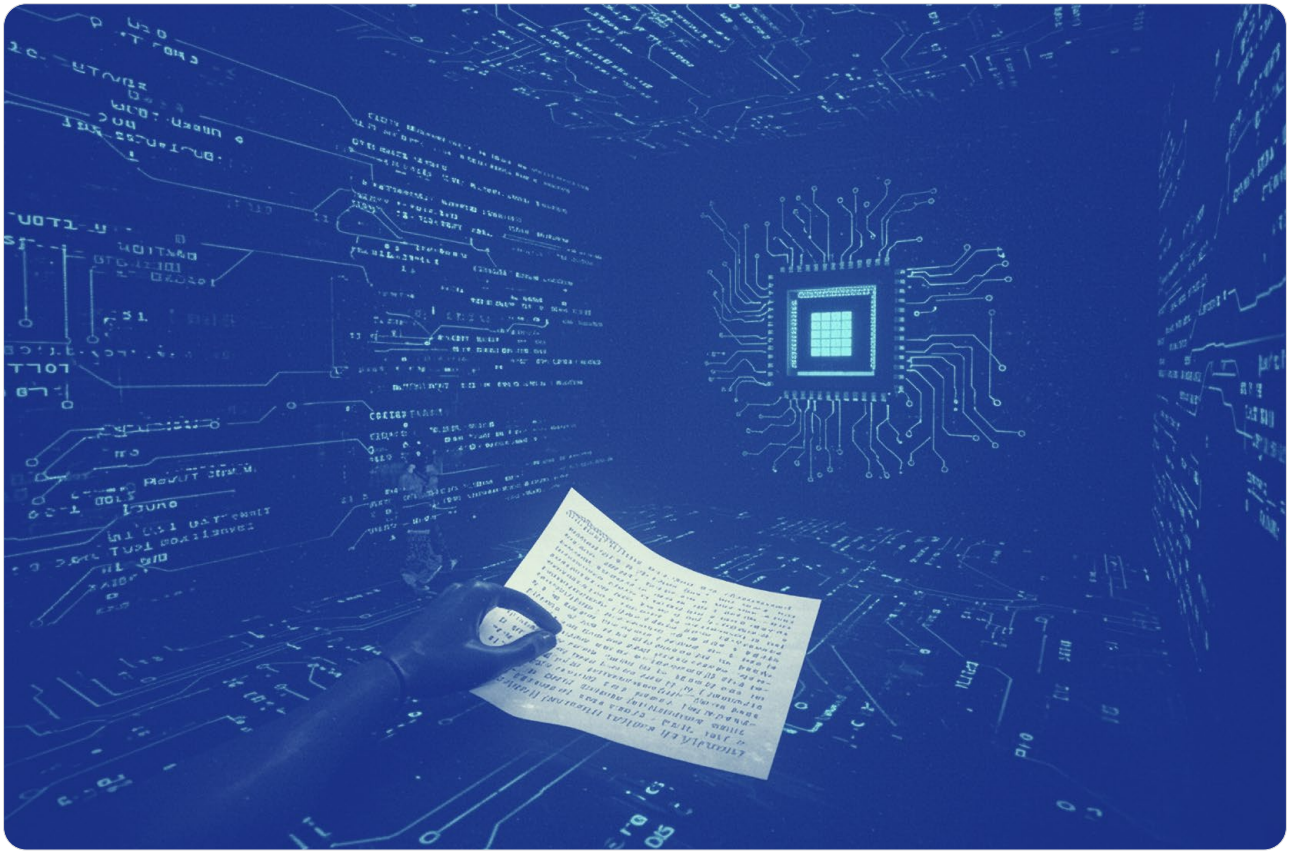
هناك أدوات متخصصة لبناء تطبيقات معتمدة على النماذج اللغوية الكبيرة، تساعد على تخصيص وتحسين المعلومات التي تولدها النماذج. من أشهرها إطار عمل (LangChain) الذي يمكّن المطورين من بناء سلاسل أوامر (Prompt Chains) جديدة أو تخصيص قوالب موجودة مسبقاً، مما يجعل هندسة الأوامر أكثر كفاءة.

أساليب كتابة الأوامر

يندرج تحت هندسة الأوامر مجموعة من الأساليب المتنوعة للتعامل مع النماذج اللغوية بما يتناسب مع المهام والأهداف المختلفة، وذلك لتحقيق أقصى استفادة من قدرات الذكاء الاصطناعي التوليدي. يستعرض هذا القسم أبرز هذه الأساليب، مع حالات استخدام مختلفة في مجال هندسة البرمجيات وأمثلة عملية توضح كيفية تطبيقها، تتضمن الأمثلة مهمة واحدة كحالة استخدام عامة "تحسين أداء تطبيق ويب". وقد صيغت هذه المهمة على النحو التالي:

"اقترح طريقة لتحسين أداء تطبيق ويب يعاني من بطء في التصفح، ثم وضِّح الأسباب المحتملة، وآثارها على تجربة المستخدم"

وفي كل أسلوب توضَّح هذه المهمة بطريقة مختلفة، بإظهار مستويات متعددة من التوجيه والتعقيد والمنهجية.



هندسة الأوامر بأمثلة قليلة (Few-Shot Prompting)

يعتمد أسلوب هندسة الأوامر بأمثلة قليلة على تضمين عدد محدود من الأمثلة داخل الأمر بهدف توجيه النموذج اللغوي نحو أداء المهمة المطلوبة. تعمل هذه الأمثلة كمرشد يوضّح للنموذج كيفية التعامل مع المهمة وتوليد الإجابة المرجوة، مما يتيح للنموذج فهم السياق، واستخلاص النمط المطلوب، ثم تعميمه وتطبيقه على مدخلات جديدة، كما أنها تمكّن النموذج من التكيّف مع مهام جديدة باستخدام عدد محدود جداً من الأمثلة، وتوجيهه لإنتاج مخرجات متسقة ووفق تنسيق محدد ومطلوب. في المقابل، يتأثر مستوى الأداء بدرجة كبيرة بجودة صياغة الأمر، إذ قد تؤدي الصياغة غير الدقيقة إلى مخرجات ضعيفة، مما يستلزم أن تكون الأمثلة المقدّمة واضحة ومعبّرة عن الهدف لضمان نتائج دقيقة ومتوافقة مع الأهداف المنشودة.

حالات الاستخدام: تصنيف تقارير الأخطاء حسب النوع، واستخراج حقول منظمة من سجلات غير منظمة، واقتراح تحسينات للكود بنفس أسلوب المشروع.

مثال

إليك بعض الأمثلة حول كيفية اقتراح حلول لمشكلات تقنية

الموضوع: ضعف أمان قاعدة البيانات

الحل المقترح: استخدام تقنيات التشفير لحماية البيانات

الموضوع: بطء تحميل الصور في تطبيق الويب

الحل المقترح: ضغط الصور وتقليل حجمها

الآن اقترح حلاً تقنياً لتطبيق ويب يعاني من بطء في التصفح

أوامر تسلسل الأفكار (Chain-of-Thought Prompting)

يعد أسلوب تسلسل الأفكار إحدى تقنيات هندسة الأوامر المصممة لتحسين أداء النماذج اللغوية في المهام التي تتطلب استدلالاً منطقياً، أو حسابياً، أو اتخاذ قرارات. يقوم هذا الأسلوب على معالجة المشكلة عبر تفكيكها إلى خطوات متتابعة، بحيث يُبنى الحل تدريجياً للوصول إلى الاستنتاج النهائي، كما يتميز بقدرته على تحسين دقة النموذج في المسائل المعقدة التي تتطلب استدلالاً متعدد الخطوات، مما يحفز النموذج لتنفيذ الحل خطوة بخطوة. ويتيح للمطوّرين تتبع سلسلة الأفكار للتحقق من صحة المنطق المتبع في كل خطوة. في المقابل، يستهلك هذا الأسلوب موارد ووقتاً أكثر مقارنةً بالإجابة المباشرة، لأن النموذج ينتج خطوات متعددة من الأفكار قبل الوصول إلى الإجابة النهائية.

لبناء أمر يعتمد على تسلسل الأفكار، يضيف المستخدم عادةً تعليمات مثل: "اشرح منطقك خطوة بخطوة" أو "وضِّح استجابتك على مراحل" في نهاية الأمر الموجَّه للنموذج اللغوي. وبذلك، لا يُطلب من النموذج إعطاء النتيجة فقط، بل أيضاً عرض الخطوات الوسيطة التي قادته للوصول إليها.

حالات الاستخدام: تحليل سجلات الخطأ، وتحديد السبب الجذري لها، وتفكيك المتطلبات المعقدة إلى مهام محددة قابلة للتنفيذ، والمقارنة بين خوارزميات مختلفة.

مثال

اقترح طريقة لتحسين أداء تطبيق ويب يعاني من بطء في التصفح، ثم وضِّح الأسباب المحتملة، وآثارها في تجربة المستخدم، اشرح استجابتك خطوة بخطوة.

أوامر تسلسل الأفكار متعددة الصيغ (Multimodal Chain of Thought)

أسلوب يحفز النموذج على الاستدلال وعرض مسار التفكير عبر صيغ متعددة، مثل الصور أو الصوتيات. على عكس النهج التقليدي لأسلوب تسلسل الأفكار الذي يركز على البُعد اللغوي فقط، يقوم النهج متعدد الصيغ على دمج المعلومات النصية والبصرية، مما يعزّز من قدرته على حل المسائل التي تشمل العناصر اللغوية والبصرية معاً، وذلك ضمن إطار عمل من مرحلتين:

- **المرحلة الأولى:** توليد المبررات استناداً إلى المعلومات المتضمنة في النص والصورة.
- **المرحلة الثانية:** استنتاج الإجابة النهائية بالاعتماد على تلك المبررات.

حالات الاستخدام: تشخيص مشاكل واجهة المستخدم من لقطات الشاشة وسجلات الخطأ، والتحقق من اتساق مخططات البيانات مع الأكواد، واستخراج بيانات منظمة من صور الجداول ووصف الأعمدة.

مثال

حلل هذا الرسم البياني الخاص بزمن استجابة الخادم وعدد الطلبات في تطبيق الويب، ثم اشرح أسباب بطء التطبيق واقترح حلولاً تقنية لتفادي هذه المشكلة.

أوامر شجرة الأفكار (Tree of Thoughts)

يشير الأسلوب إلى تحفيز النموذج نحو استكشاف فروع متعددة من الاستدلال أو الأفكار قبل الوصول إلى المخرج النهائي. وهو إطار عام يتجاوز أسلوب سلسلة الأفكار، وذلك عبر النظر في مسارات تفكير متعددة وتقييمها ذاتياً بما يقود إلى الحل الأمثل. يمكن تمثيله بهيكل شجري يبدأ بالمشكلة الأساسية، ثم يولد خطوات متعددة على شكل "أفكار" كما في أسلوب تسلسل الأفكار. يتم بعد ذلك تقييم كل خطوة ومدى تقدمها نحو الحل، ثم يقع الاختيار على الخطوات المتقدمة ومتابعتها في المخرج النهائي. كما يمكّن هذا الأسلوب النموذج من التراجع عند التنبؤ بأن المسار المختار لن يؤدي إلى حل جيد. يعزّز ذلك من قدرة النموذج على حل المشكلات المعقدة عبر استكشاف مسارات تفكير متعددة بشكل متفرّع بدلاً من اتباع سلسلة وحيدة من الأفكار. يحاكي هذا الأسلوب طريقة تفكير البشر في تجربة حلول مختلفة ثم اختيار الأنسب، وقد أثبت نجاحه في مهام تتطلب التخطيط والتفكير الاستراتيجي بمعدل نجاح أعلى من الطرق التقليدية. كما يسمح بالتعامل مع حالة عدم اليقين (uncertainty) أثناء الاستدلال، وذلك لتقدير مدى موثوقية كل مسار، مما يتيح نتائج أكثر دقة واعتمادية في المواقف الحساسة. في المقابل، يتطلب هذا الأسلوب قدراً كبيراً من الموارد الحسابية، نظراً إلى حاجته إلى تتبع فروع متعددة من الحلول المحتملة والقدرة على التراجع واستكشاف بدائل عدة أثناء البحث عن الحل.

حالات الاستخدام: تصحيح الأخطاء المعقّدة عبر توليد فرضيات متعددة وتتبع آثارها، واقتراح تحسينات على الكود عبر تحليل التأثير في مسارات مختلفة.

مثال

اقترح ثلاث طرق ممكنة لتحسين أداء تطبيق ويب بطيء. اشرح آلية العمل والمزايا والعيوب لكل طريقة.
بعد ذلك، اختر الطريقة الأكثر فعالية، ووضح سبب اختيارك.

أوامر الاتساق الذاتي (Self-Consistency prompting)

يُعد هذا الأسلوب امتداداً لمنهجية تسلسل الأفكار، حيث يقوم على توليد عدة مسارات استدلالية مستقلة ومختلفة من النموذج، ثم اختيار النتيجة الأكثر شيوعاً واتساقاً لاعتمادها كمخرج نهائي. يساهم هذا النهج في رفع مستوى الموثوقية والدقة، عبر الاعتماد على مبدأ الإجماع المبني على متوسط استجابات متعددة لنفس الأمر. أدى ذلك إلى تحسين الأداء مقارنة بأسلوب تسلسل الأفكار التقليدي الذي يعتمد على مسار تفكير واحد، لأن النموذج يتجنب الأخطاء الحسابية الشائعة من خلال المقارنة والتصويت بين حلول متعددة. يواجه هذا الأسلوب تحديات في الاستهلاك المرتفع للموارد الحاسوبية والزمنية مقارنة بتوليد إجابة واحدة، نظراً إلى حاجة النموذج إلى إنتاج عدة مسارات من الأفكار المستقلة لنفس السؤال.

حالات الاستخدام: التخطيط التسلسلي للمهام المعقدة، وتوليد فرضيات متعددة لمسار الخطأ ثم تحديد السيناريو الأرجح، وتحليل المخاطر الأمنية المحتملة وترتيبها حسب الاتساق مع بنية النظام.

مثال

بصفتك مطوِّر برمجيات يهتم بتحسين تجربة المستخدم، وضِّح الأسباب الشائعة التي تؤدي إلى بطء التصفح في تطبيقات الويب، ثم اقترح حلولاً عملية قابلة للتنفيذ، مثل استخدام تقنيات التخزين المؤقت، أو تطبيق استراتيجيات التحميل البطيء (Lazy Loading) أو تحسين أداء جافا سكريبت.

أوامر تحفيزية موجهة (Directional Stimulus Prompting - DSP)

يعتمد هذا الأسلوب على إشارات توجيهية لدفع النموذج اللغوي نحو توليد نوع محدد من الاستجابات المطلوبة، مما يساعد على ضبط منظور أو نبرة الاستجابة. كما يتميز بالقدرة على تحسين الدقة والكفاءة عبر التركيز على العناصر المهمة داخل النص، مثل تحديد الكلمات المفتاحية لمقالة في مهام التلخيص. يتطلب الأسلوب إعداداً دقيقاً لضمان فعالية التوجيهات، في المقابل يعزز موثوقية المخرجات في المهام التي تتطلب مراعاة دقيقة للسياق. كما يُقلل من حجم البيانات المطلوبة ويختصر زمن المعالجة عبر التركيز على "المحفزات" المهمة بدلاً من إدخال كم كبير من السياقات غير الضرورية. يؤدي ذلك إلى تكلفة حاسوبية أقل للحصول على نتائج جيدة، دون التقليل من جودة المخرجات.

حالات الاستخدام: توليد كود ملتزم بقيود صريحة، وفحص الكود البرمجي من ناحية استهلاك الذاكرة وتقديم توصيات قابلة للتنفيذ، وتوليد سيناريوهات استخدام للتطبيق وفق شخصيات محددة وقيود واضحة.

مثال

قدم ثلاثة تفسيرات مختلفة لمشكلة بطء التصفح في تطبيقات الويب، واقترح الحلول الممكنة وآلية تطبيقها. ثم حدد الحل الأكثر دقة وقابلية لتحسين تجربة المستخدم.

التوليد المعزز بالاسترجاع (Retrieval Augmented Generation – RAG)

يجمع هذا الأسلوب بين استرجاع المعلومات من مصادر خارجية وقدرات النماذج اللغوية التوليدية على إنتاج استجابات تعتمد على معرفة حديثة أو متخصصة في مجال معين. يتيح ذلك اتساقاً أكبر في المعلومات الواقعية، ويحسن موثوقية الاستجابات المولَّدة، ويساعد على التخفيف من مشكلات الهلوسة أو الاستجابات المضللة. يقوم هذا الأسلوب على إرسال مدخل محدد واسترجاع مجموعة من المستندات ذات الصلة من مصدر معيّن (مثل ويكيبيديا). ثم دمج هذه المستندات ضمن السياق مع الأمر الأصلي وتمريرها إلى النموذج اللغوي لإنتاج المخرج النهائي. تعتمد جودة المخرجات بشكل مباشر على جودة وملاءمة البيانات الخارجية المسترجعة. إذا كان مصدر المعرفة يحتوي على معلومات قديمة أو غير دقيقة أو منحازة، سينعكس ذلك سلباً على إجابات النموذج. لذا، يتطلب ذلك توفير مصادر موثوقة لضمان دقة المعلومات المستخدمة.

حالات الاستخدام: الحصول على ردود فنية مدعومة بمقالات من قاعدة المعرفة والتذاكر القديمة وسجلات الأنظمة، والتأكد من توافق الرخص مفتوحة المصدر مع سياسات المؤسسة، والإعداد السريع للمطورين الجدد عبر شرح هيكلية البيانات وخطوات البناء والتشغيل.

مثال

باستخدام بيانات وإرشادات تحسين الأداء من (Google Analytics 4)، اقترح خطة لمعالجة بطء التصفح في تطبيق ويب، مع تحديد الأسباب الأكثر شيوعاً، وشرح آثارها في تجربة المستخدم.

أفضل الممارسات لبناء الأوامر

يقدم هذا القسم خلاصة إرشادات عملية لهندسة الأوامر الموجهة إلى المطورين والمبرمجين لتحسين فعالية الأوامر الموجهة للنماذج اللغوية، وضمان مخرجات أكثر دقة واتساقاً وقابلة للتنفيذ في البيئات البرمجية.

تقديم تعليمات واضحة ومحددة

يعد توجيه النموذج بتعليمات واضحة ومحددة وسيلة فعَّالة لتخصيص سلوك النموذج وتحسين مخرجاته. لذلك، ينبغي أن تكون التعليمات واضحة ومباشرة وموجزة، وأن تتضمن المعلومات المطلوبة وشكل المخرجات المرغوب.

✗ **أمر عام وغير فعَّال**

الأمر

حلل سجل الخادم التالي:

“ERROR /api/v1/login TimeoutException user_id=1021 08:25:41 2025-10-13”

الاستجابة

هذا خطأ في تسجيل الدخول.

ملاحظات

هـذا الأمـر غامـض ولا يوضّـح المطلـوب بدقـة؛ فهـو لا يحـدد نـوع المعلومـات المـراد اسـتخراجها أو طريقـة عرضهـا، ممـا يـؤدي إلـى اسـتجابات سـطحية وغيـر منظمـة.

✓ **أمر واضح ودقيق**

الأمر

استخرج من السجل التالي الحقول المحددة:

السـجل: “ERROR /api/v1/login TimeoutException user_ 08:25:41 2025-10-13 id=1021“

الحقول:

(timestamp, level, endpoint, error type, user_id)

الاستجابة

```
{
  "timestamp": "٠٨:٢٥:٤١ ١٣-١٠-٢٠٢٥",
  "level": "ERROR",
  "endpoint": "/api/v١/login",
  "error_type": "TimeoutException",
  "user_id": ١٠٢١
}
```

ملاحظات

هـذا الأمـر فعّـال ويوضـح المطلـوب بدقـة، إذ يوجّـه النمـوذج لاسـتخراج حقـول محـددة وإخراجهـا بصيغـة منظمـة (JSON)، ممـا ينتـج اسـتجابات دقيقـة وقابلـة للاسـتخدام البرمجـي.

إضافة المعلومات السياقية

تضمين جميع التعليمات والمعلومات التي يحتاجها النموذج إلى حل المشكلة، مما يساعد النموذج على فهم القيود والتفاصيل الخاصة بالمهمة المطلوبة وتنفيذها بأفضل طريقة ممكنة. وتشمل المعلومات السياقية الفعّالة ما يلي:

- **المعلومات الأساسية** التي يمكن للنموذج الرجوع إليها أثناء توليد الاستجابات.
- **القواعد أو القيود** التي توجِّه سلوك النموذج وتتحكم في طريقة تفاعله.

✗ **أمر عام بدون معلومات سياقية**

الأمر

أنشئ كود (Python) لبناء واجهة إدخال بيانات.

الاستجابة

```
name = input(“Enter your name: “)
age = input(“Enter your age: “ )
print(“Hello” , name(
```

ملاحظات

لم يتم تحديد نوع الواجهة ولا الغرض من الواجهة أو الأداة المراد استخدامها.

✓ **أمر محدد يتضمن معلومات سياقية**

الأمر

أنشئ كود (Python) لبناء واجهة إدخال بيانات مستخدم باستخدام مكتبة (Streamlit).
يجب أن تحتوي الواجهة على حقول لإدخال الاسم، والعمر، والبريد الإلكتروني، وزر ”إرسال“.
عند الضغط على الزر، تُعرض البيانات في صندوق منسق.
استخدم ألواناً محايدة، وتأكد أن التصميم متجاوب مع مختلف أحجام الشاشات.

الاستجابة

```
import streamlit as st
st.title("نموذج إدخال بيانات المستخدم")
name = st.text_input("الاسم:")
age = st.number_input("العمر:", min_value=١, max_value=١٠٠)
email = st.text_input("البريد الإلكتروني:")
if st.button("إرسال"):
    st.success(f"الاسم: {name}\nالعمر: {age}\nالبريد الإلكتروني:
{email}")
```

ملاحظات

هذا الأمر واضح، حدَّد المكتبة المستخدمة (Streamlit)، والغرض الوظيفي للواجهة، والتصميم العام لها، مما وجَّه النموذج لتوليد كود مناسب وواضح الهدف.

تنظيم وهيكلة الأوامر

تؤثر طريقة تنظيم وهيكلة الأوامر في قدرة النموذج على فهم المعلومات المضمنة وتفسيرها بدقة. كما يساعد النموذج على فهم العلاقات بين أجزاء المعلومات واستخدامها بطريقة صحيحة. ومن أبرز الممارسات في هيكلة الأوامر:

- **استخدام المقدمات التوضيحية (Prefixes) مع الأوامر البسيطة:** تشير المقدمة إلى كلمة أو عبارة متبوعة بنقطتين رأسيتين تُستخدم لتوسيم نوع المعلومة داخل الأمر.

مثال: يمكن استخدام مقدمات مثل (TASK) أو (CLASSES) أو (OBJECTS) لتوضيح مكوّنات مختلفة داخل النص.

- **استخدام وسوم (XML) في الأوامر المعقدة:** يُوصى باستخدام وسوم (XML) في الأوامر المعقدة أو محددات بصيغ أخرى لفصل مكونات الأمر بشكل واضح.

مثال: يمكن الاستعانة بعبارات مثل (BEGIN) أو (END) أو الأقواس{} لتحديد الأقسام المختلفة داخل الأمر الطويل، مما يساعد النموذج على تمييز التعليمات الفعلية عن المحتوى الإضافي بطريقة أكثر دقة ووضوحاً.

✕ **أمر غامض وغير منظّم**

الأمر حوّل هذا الكود من صيغة (JSON) إلى (TypeScript)

```
{
  “userId”: 1021,
  “username”: “Ahmad”,
  “isActive”: true,
  “roles”: [“admin”, “editor”]
}
```

الاستجابة كود (TypeScript):

```
const data = { userId: 1021, username: “Ahmad”, isActive:
true, roles: [“admin”,”editor”] };
```

ملاحظات لم يوضّح بدقة المطلوب من التحويل، هل هو إنشاء واجهة (Interface) أو شيء (Object) أو نموذج بيانات (Model)؟

ونتيجة لذلك، قدّم النموذج مخرجات غير مناسبة للغرض البرمجي المطلوب.

✓ **صياغة أفضل باستخدام المقدمات التوضيحية**

الأمر

TASK:

أنشئ تعريف واجهة (Interface) بلغة (TypeScript) بناءً على الشيء التالي:

JSON:

```
{
  “userId”: 1021,
  “username”: “Ahmad”,
  “isActive”: true,
  “roles”: [“admin”, “editor”]
}
```

الاستجابة

```typescript
interface User {
  userId: number;
  username: string;
  isActive: boolean;
  roles: string[];
}
```

ملاحظات

هذا الأمر فعّال ويوضح المطلوب بدقة، إذ يوجّه النموذج لاستخراج حقول محددة وإخراجها بصيغة منظمة (JSON)، مما ينتج استجابات دقيقة وقابلة للاستخدام البرمجي.

تقسيم المهام المعقدة إلى أوامر أبسط

تتطلب المهام المعقدة تعليمات متعددة وخطوات تفصيلية. ولتحسين استجابات النموذج مع هذه الحالات، يُنصح بتقسيم الأمر إلى مهام فرعية صغرى. تساعد هذه الأوامر الفرعية على زيادة التحكم، وتسهيل تصحيح الأخطاء، وتحسين الدقة. ومن أبرز الطرق في تقسيم الأوامر المعقدة:

- **تسلسل الأوامر (Chain Prompt):** تقسيم المهمة إلى مهام فرعية، وإرسالها إلى النموذج بشكل متتابع. يُنصح باستخدامها في المهام المعقدة التي تتضمن خطوات متتابعة، فتصبح كل خطوة أمراً منفصلاً، وتترابط الأوامر معاً في تسلسل منطقي. في هذا التسلسل، يُستخدم الأمر الأول كمدخل للأمر التالي، ويُعد مخرج الأمر الأخير هو الناتج النهائي.
- **تجميع الاستجابات (Aggregate Responses):** تقسيم المهمة إلى مهام فرعية، وإرسالها إلى النموذج بشكل متوازٍ. يُنصح باستخدامها في المهام المعقدة التي لا يتطلب لتنفيذها ترتيب محدد، وذلك من خلال إرسال الأوامر بشكل منفصل ثم تجميع استجابات النموذج في أمر نهائي للحصول على نتيجة واحدة شاملة.

تحليل سلوك الشراء في متجر إلكتروني لتحديد المنتجات الأنسب للعروض الترويجية القادمة، بالاعتماد على بيانات المبيعات وبيانات التفاعل مع المنتجات.

المهمة الأولى	تحليل مجموعتي البيانات: بيانات المبيعات وبيانات المنتجات المفضلة. وذلك من خلال تقسيم المهمة إلى مهمتين، ثم إرسال الأوامر الخاصة بكل مهمة بشكل متوازٍ.

المهمة (A-1)

تحليل بيانات المبيعات

الأمر	أنشئ شريط تنقل (Navigation Bar) في أعلى الصفحة يحتوي على شعار في الجهة اليسرى وثلاثة روابط في الجهة اليمنى، ويكون التصميم متجاوباً مع الشاشات الصغيرة.
المدخلات	بيانات الشراء (Store_Sale_Data).
النتيجة المتوقعة	يُتوقع أن تحتوي النتيجة على عدد المبيعات لكل منتج خلال الربع الأخير، منسقاً بصيغة (JSON).

المهمة (1-B)	**تحليل سلوك التصفح**
	الأمر: حدّد المنتجات التي تمت إضافتها إلى عربة التسوق أو تم الاطلاع عليها بشكل متكرر دون إتمام عملية الشراء. يرجى تنظيم النتائج بصيغة (JSON) بحيث يكون اسم المنتج هو المفتاح (Key)، وعدد مرات مشاهدتها هو القيمة (Value). **المدخلات**: بيانات المنتجات (Products_Data). **النتيجة المتوقعة**: يُتوقع أن تحتوي النتيجة على عدد المنتجات التي حظيت باهتمام المستخدمين منسقاً بصيغة (JSON).
المهمة الثانية	**تجميع البيانات: بيانات المنتجات الأعلى مبيعاً وبيانات المنتجات الأكثر تفاعلاً، ثم تحليلها لإنشاء قائمة بالعروض المقترحة.**
	الأمر: اجمع بيانات المبيعات وبيانات المنتجات التي حصلت على اهتمام المستخدمين، ثم أنشئ قائمة بالعروض المقترحة، بحيث تمثل 75% من المنتجات الأعلى مبيعاً، و 25% من المنتجات الأكثر اهتماماً من المستخدمين. **المدخلات**: (TASK_1A_RESPONSE) و (TASK_1B_RESPONSE). **النتيجة المتوقعة**: يُتوقع أن يحتوي المخرج على قائمة بالعروض الترويجية للمنتجات، تتضمن مزيجاً من أكثر المنتجات مبيعاً والمنتجات التي أظهرت تفاعلاً مرتفعاً من المستخدمين.

التعديل على معاملات النموذج

يتضمن كل طلب يُرسل إلى النموذج قيم معاملات (Parameters) تتحكم في كيفية توليد الاستجابة. يُمكن أن ينتج النموذج مخرجات مختلفة بناءً على اختلاف هذه القيم، لذا يُنصح بتجربة إعدادات متعددة للوصول إلى أفضل أداء للمهمة المطلوبة. قد تختلف المعاملات المتاحة من نموذج إلى آخر، لكن أكثرها شيوعاً هي:

- **عدد الوحدات القصوى (Max Output Tokens):** هي الحد الأقصى لعدد الوحدات (Tokens) التي يمكن للنموذج توليدها في الاستجابة. تُستخدم القيمة المنخفضة للحصول على استجابات أقصر، والقيمة الأعلى للحصول على استجابات أطول.

- **معامل حرارة النموذج (Model Temperature):** يُحدد معامل الحرارة مدى تنوع المخرجات، فالقيم المنخفضة (مثل: 0 – 0.3) مناسبة للأوامر التي تتطلب استجابات دقيقة ومحددة، والقيم الأعلى (مثل: 1.0 – 2.0) تناسب المخرجات الإبداعية والمتنوعة. بينما القيمة (0) فتعني أن النموذج يختار الوحدات ذات الاحتمالية الأعلى (نتائج شبه ثابتة).

- **أخذ عينات أعلى (ب) (Top-P Sampling):** يحدد هذا المعامل كيف يختار النموذج الوحدات التالية بناءً على مجموعة من الاحتمالات التراكمية. يختار النموذج الوحدات الأعلى احتمالاً إلى الأقل حتى يصل مجموع احتمالاتها إلى الحد الأعلى وهو قيمة (Top-P). على سبيل المثال، إذا كانت احتمالات الوحدات (A=0.3, B=0.2, C=0.1) وكان (Top-P=0.5)، فسيختار النموذج A أو B فقط ويستبعد C. لذلك، تُستخدم قيمة منخفضة لهذا المعامل للحصول على مخرجات أكثر اتساقاً، وقيمة أعلى للحصول على مخرجات أكثر عشوائية وتنوعاً.


33

هندسة السياق

ظهر مصطلح هندسة السياق (Context Engineering) باعتباره إطاراً أوسع وأكثر شمولية من هندسة الأوامر، حيث يركز على تصميم وإدارة تدفق المعلومات الداخلة إلى أنظمة الذكاء الاصطناعي والخارجة منها. فبينما تركز هندسة الأوامر على صياغة مجموعة من التعليمات في نص واحد، فإن هندسة السياق تتجاوز ذلك نحو أساليب أكثر شمولية لاستخراج المعرفة من مصادر متعددة وتنظيمها داخل نافذة السياق.

تُعنى هندسة السياق **بتصميم وبناء أنظمة ديناميكية توفر المعلومات والأدوات المناسبة، بالصيغة المناسبة، وفي الوقت المناسب، لتمكين النموذج اللغوي من إنجاز المهمة المطلوبة بكفاءة.**

ولفهم هندسة السياق، من الضروري إعادة صياغة مفهوم "السياق" بشكل موسَّع يتجاوز التصور التقليدي باعتباره مجرد أمر (Prompt) يُقدَّم إلى النموذج اللغوي. فالسياق يشمل جميع العناصر والمعلومات التي يحصل عليها النموذج قبل توليد الاستجابة، كما هو موضح في **الشكل (11)**.

الشكل (1): أبرز العناصر التي يتضمنها السياق

1أبرز عناصر السياق

أوامر النظام (System Prompt)

مجموعة أولية من التعليمات تحدد سلوك النموذج أثناء المحادثة، ويمكن أن تتضمن أمثلة أو قواعد وغير ذلك.

أوامر المستخدم (User Prompt)

المهمة أو السؤال المباشر القادم من المستخدم.

الذاكرة قصيرة المدى (Short-Term Memory)

المحادثة الحالية، بما في ذلك استجابات المستخدم والنموذج.

الذاكرة طويلة المدى (Long-Term Memory)

قاعدة معرفة مستمرة يتم جمعها عبر عدد من المحادثات السابقة، تشمل تفضيلات المستخدم التي تعلمها النموذج وملخصات المشاريع السابقة وغيرها من المعلومات المكتسبة.

المعلومات المسترجعة من أساليب التوليد المعزز بالاسترجاع (RAG)

المعرفة المسترجعة من مصادر خارجية، مثل المعلومات ذات الصلة من المستندات أو قواعد البيانات أو الواجهات البرمجية (APIs).

الأدوات المتاحة (Available Tools)

جميع الدوال أو الأدوات المتاحة التي يمكن للنموذج استدعاؤها (مثل: check_inventory، send_email).

المخرجات المهيكلة (Structured Output)

تعريف بالصيغة المرغوبة لاستجابة النموذج (مثل: JSON، Markdown).

خصائص هندسة السياق

- **نظام متكامل:** تشير هندسة السياق إلى بُنية شاملة تُنفذ قبل استدعاء النموذج الرئيسي، وليست مجرد أمر أو قالب أمر ثابت.
- **ديناميكية:** تتميز هندسة السياق بالقدرة على التكيف اللحظي بما يتناسب مع طبيعة المهمة المباشرة ومتطلباتها.
- **المعلومات والأدوات المناسبة:** تعتمد هندسة السياق على توفر المعرفة (المعلومات) والقدرات (الأدوات) في الوقت الصحيح، وذلك لضمان ألا يفتقر النموذج إلى التفاصيل الأساسية اللازمة.
- **أهمية الصياغة:** تؤثر طريقة عرض المعلومات بشكل مباشر في كفاءة الأداء. فالمُلخص الموجز أفضل من إلقاء كم ضخم من البيانات غير المعالجة، كما أن المخطط الواضح للأداة أفضل من التعليمات الغامضة.

ومن هذا المنطلق، لم يعد بناء وكلاء ذكاء اصطناعي فعّالين وموثوقين يتمحور حول إيجاد أمر مثالي أو الاعتماد على تحديثات النماذج فقط، بل أصبح يعتمد بدرجة أكبر على هندسة السياق وتوفير المعلومات الصحيحة والأدوات المناسبة بالشكل الصحيح وفي الوقت المناسب. ويمثل ذلك تحدياً متعدد الجوانب يتطلب فهماً دقيقاً لحالة الاستخدام، وتحديد المخرجات المرجوة، وتنظيم المعلومات الأساسية بصورة منهجية تمكّن النموذج اللغوي من أداء مهامه بكفاءة وفعالية.

التحديات والأبعاد الأخلاقية

يتناول هذا القسم أبرز التحديات والأبعاد الأخلاقية المرتبطة باستخدام النماذج اللغوية وهندسة الأوامر، مع تسليط الضوء على مخاطر الأوامر المضللة، وحماية البيانات والملكية الفكرية، وأهمية الالتزام بمبادئ الاستخدام المسؤول لضمان تطبيق آمن وموثوق للتقنيات الذكية.

مخاطر الأوامر المضللة

تشير الأوامر المضللة إلى التعليمات المصاغة -عمداً أو عن غير قصد- بطريقة تدفع النموذج اللغوي إلى إنتاج مخرجات غير آمنة، أو مضللة، أو مخالفة للسياسات. وقد تؤدي هذه الأوامر إلى تجاوز الضوابط أو تضمين محتوى لا يتوافق مع المعايير الأخلاقية والأمنية المعتمدة. لذلك، ينبغي على المطورين التحقق من سلامة الأوامر والمخرجات قبل اعتمادها في البيئات التشغيلية، وضمان تطبيق آليات المراجعة والتحقق والفلترة الأمنية ضمن مراحل التطوير. كما يُوصى بأن تعمل المؤسسات على تأهيل فرق العمل وتوعيتها بمخاطر الأوامر المضللة وآليات الحد منها، لضمان استخدام النماذج التوليدية بصورة مسؤولة وآمنة ومتوافقة مع السياسات الوطنية للذكاء الاصطناعي.

الملكية الفكرية وحماية البيانات

قد تُنتج النماذج اللغوية التوليدية مخرجات تتضمن بيانات شخصية أو معلومات حساسة، أو أعمالاً محمية بموجب حقوق الملكية الفكرية. لذا، يتعيّن على المطورين التأكد من أن المحتوى الناتج لا ينتهك حقوق النشر أو الاستخدام، وأنه يتوافق مع أنظمة حماية البيانات والخصوصية المعمول بها محلياً ودولياً. كما ينبغي الالتزام بمبدأ الحد الأدنى من البيانات (Data Minimization) عند تصميم الأوامر، وتجنّب إدخال بيانات سرية أو معلومات تخص الأفراد أو المؤسسات. يجب كذلك على المؤسسات وضع سياسات واضحة لتخزين البيانات ومشاركتها، وتدريب فرق التطوير على أفضل ممارسات الخصوصية لضمان الامتثال للأنظمة وتعزيز ثقة المستخدمين.

مبادئ الاستخدام المسؤول

تؤكد مبادئ استخدام الذكاء الاصطناعي الصادرة من الهيئة السعودية للبيانات والذكاء الاصطناعي (سدايا) على ضرورة تبني مبادئ الاستخدام المسؤول، بما في ذلك النزاهة والإنصاف، والموثوقية، والشفافية، وقابلية التفسير. كما ينبغي على المطورين تطبيق هذه المبادئ عند صياغة الأوامر، لضمان مخرجات عادلة ودقيقة وآمنة، ولا تؤدي إلى أضرار أو محتوى مضلل.

المراجع

- Google Cloud, Introduction to prompting
- Llama, Prompting Guide
- OpenAI, Prompt Engineering
- Anthropic, Prompt Engineering Overview
- IBM, Prompt Engineering Techniques
- Microsoft, Prompt Engineering Concepts
- Prompting Guide, The Complete Guide to Prompt Engineering for AI Models

للاستفسارات:

Studies@sdaia.gov.sa

نأمل مشاركتنا رأيك حول هذه الدراسة من خلال مسح الرمز لتعبئة الاستبيان.

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

AITTA
مركز دراسات الذكاء الاصطناعي
Artificial Intelligence Think Tank

SDAIA.GOV.SA SDAIA_SA SDAIA.SAUDI SDAIA-KSA

يناير 2026
نوع الوثيقة | أدلة إرشادية
تصنيف الوثيقة | عام
رقم الوثيقة | SDAIA-P136
رقم الإصدار | 1.0