دليل تأسيسي

ما لا يسعك جهله عن الذكاء الاصطناعي

من النموذج إلى الوكيل: خريطة مبسّطة لفهم الأدوات والسياق وبيئات التشغيل

النموذج	الأدوات	الطبقة التشغيلية	الوكيل
العقل الذي يعمل بالكلام	كيف يتحول الكلام إلى فعل؟	لماذا يبدو النموذج مختلفاً؟	حين يعمل النظام في حلقة

م. علاء أبو جيش

ما لا يسعك جهله عن الذكاء الاصطناعي

من النموذج إلى الوكيل: خريطة مبسّطة لفهم الأدوات والسياق وبيئات التشغيل

كل يوم يظهر نموذج جديد، وأداة جديدة، واسم جديد يَعِدُك بقدرات لم تكن ممكنة بالأمس. تدخل إلى ChatGPT، ثم تسمع عن Claude، ثم ترى شخصاً يعمل داخل Claude Code، وآخر يبني باستخدام Codex، وثالثاً يتحدث عن Agents وSkills وMCP وAPIs وكأنها كلمات يفترض أن يفهمها الجميع.

وسط هذا كله، من الطبيعي جداً أن تشعر بأنك وصلت متأخراً إلى عالم يتحرك أسرع من قدرتك على اللحاق به.

لكن المفاجأة أن هذا العالم أقل تعقيداً مما يبدو.

كثرة الأسماء لا تعني كثرة المبادئ. معظم أدوات النماذج والوكلاء الحديثة تعيد تركيب مجموعة صغيرة من اللبنات الأساسية: **نموذج، وسياق، وأدوات، وبيئة تشغيل، وصلاحيات، وحلقة عمل**. ما إن تفهم هذه اللبنات والعلاقة بينها، حتى تتوقف الأدوات عن الظهور لك كصناديق سحرية منفصلة. ستبدأ برؤية الهيكل نفسه خلفها، مهما تغيّرت الأسماء والواجهات.

هذا الدليل كُتب لهذه الغاية.

لن يعلّمك استخدام أداة بعينها خطوة بخطوة، ولن يغرقك في الرياضيات التي بُنيت بها النماذج. سنفعل شيئاً أبسط: سنفتح الغطاء، ونرى كيف تعمل المنظومة من الداخل بلغة يستطيع فهمها شخص غير تقني.

حين تنتهي منه، ستكون أكفأ في استخدام الذكاء الاصطناعي، وأسرع في الانتقال بين الأدوات من دون أن تبدأ من الصفر كل مرة. ستفهم لماذا يبدو Claude Code مختلفاً عن Claude، ولماذا يستطيع Codex فعل أشياء لا تفعلها محادثة عادية، ولماذا ينسى النموذج، وكيف يتحول كلامك إلى فعل حقيقي على الكمبيوتر أو في خدمة خارجية.

وسيفيدك هذا الفهم حين تبدأ ببناء تطبيق، أو عملية أتمتة، أو وكيل خاص بعملك. عندما تظهر مشكلة، لن يكون كل ما أمامك صندوقاً غامضاً؛ ستعرف أي جزء من المنظومة يحتاج إلى إصلاح.

اقرأ الدليل مرة لفهم القصة، لا لحفظ الكلمات. ثم ارجع إلى فهرس المصطلحات والقاموس كلما قابلت اسماً جديداً.

فهرس الدليل

المصطلحات التي سنفككها

هذه ليست تعريفات مطلوباً منك حفظها الآن. هي خريطة لما سنصل إليه، وسبب حاجتك إلى كل كلمة:

المصطلح	بالإنجليزية	لماذا ستحتاج إلى فهمه؟
النموذج	Model	لتعرف أين يعيش الذكاء نفسه، وما الذي يستطيع فعله وحده وما الذي لا يستطيع.
التوكن	Token	لتفهم كيف يدخل كلامك إلى النموذج، وكيف تُقاس السعة والتكلفة.
الطلب	Prompt	لتفصل بين ما تطلبه أنت وبين كل المعلومات الأخرى التي تصل إلى النموذج.
السياق	Context	لتعرف ما الذي يراه النموذج فعلياً في هذه اللحظة.
نافذة السياق	Context Window	لتفهم لماذا للمحادثة ذاكرة محدودة، ولماذا يتراجع الأداء أحياناً مع طولها.
الأداة	Tool	لتفهم كيف يستطيع النموذج أن يبحث أو يكتب ملفاً أو يرسل بريداً.

المصطلح	بالإنجليزية	لماذا ستحتاج إلى فهمه؟
استدعاء الأدوات	Tool Calling	لتفهم كيف يطلب النموذج إجراءً ثم يستقبل نتيجته.
الطبقة التشغيلية	Harness	لتفهم لماذا تختلف قدرة النموذج نفسه من منتج إلى آخر.
الوكيل	Agent	لتفهم كيف ينتقل النظام من إجابة واحدة إلى تنفيذ مهمة متعددة الخطوات.
الحلقة الوكيلية	Agentic Loop	لتفهم كيف ينفذ الوكيل خطوة، ويرى النتيجة، ثم يقرر ما بعدها.
تعليمات النظام	System Prompt	لتعرف ما الذي قرأه النموذج عن دوره وقوانينه قبل رسالتك الأولى.
الذاكرة والحالة	Memory & State	لتفهم ما الذي يبقى بين الجلسات، وكيف يُستأنف العمل.
المهارة	Skill	لتمنح الوكيل طريقتك في إنجاز عمل يتكرر.
واجهة البرمجة ومفتاحها	API & API Key	لتوصل الوكيل بخدمة أو قدرة غير موجودة لديه.
بروتوكول MCP	Model Context Protocol	لتربط الوكيل بأدوات وبيانات خارجية بطريقة موحدة.
قاعدة البيانات	Database	لتعرف أين تعيش بيانات العمل التي يمكن الرجوع إليها وتحديثها.
الوكيل الفرعي	Sub-agent	لتفهم كيف يفوّض الوكيل جزءاً من العمل إلى عامل مستقل.
الأتمتة والتنسيق	Automation & Orchestration	لتحول التنفيذ من طلب يدوي إلى عملية متكررة ومنظمة.

1 أول سبب للفوضى: نحن نقارن أشياء من طبقات مختلفة

قبل أن نسأل: «أيهما أفضل، Claude أم ChatGPT أم Codex؟» علينا أن نسأل سؤالاً أبسط:

ما نوع الشيء الذي نتحدث عنه أصلاً؟

في سوق الذكاء الاصطناعي تُستخدم الأسماء بطريقة مربكة. قد يكون الاسم اسم شركة، أو عائلة نماذج، أو منتجاً كاملاً، أو واجهة محادثة، أو بيئة وكلاء. وأحياناً يُستخدم الاسم نفسه لأكثر من طبقة.

هذه أهم الطبقات:

الطبقة	ما هي؟	أمثلة شائعة
الشركة أو المزوّد	الجهة التي تبني النماذج أو المنتجات وتدير البنية التحتية	OpenAI، Anthropic، Google
النموذج Model	النظام المدرّب الذي يستقبل مدخلات ويولّد مخرجات	عائلات GPT، Claude، Gemini
المنتج أو الواجهة	الشيء الذي يفتحه المستخدم ويتعامل معه	موقع محادثة، تطبيق مكتبي، إضافة داخل محرر
الطبقة التشغيلية Harness	النظام الذي يضع النموذج داخل منتج، ويحدد ما يراه وما يستطيع استخدامه	الطبقة الموجودة داخل ChatGPT أو Claude Code أو Codex، أو طبقة تبنيها أنت
بيئة العمل Environment	المكان الذي يحدث فيه العمل فعلياً	محادثة في المتصفح، مجلد مشروع على الكمبيوتر، أو خادم على الإنترنت
الأداة Tool	قدرة تنفيذية محددة متاحة داخل البيئة	البحث، قراءة ملف، تشغيل أمر، إرسال بريد
الوكيل Agent	النظام أثناء عمله لتحقيق هدف عبر عدة خطوات وأدوات	وكيل بحث، وكيل برمجة، وكيل يدير مهمة متكررة

قد يحمل المنتج اسم النموذج نفسه، وقد يتيح لك أكثر من نموذج. لذلك لا تحفظ الأسماء وحدها؛ تعلّم أن تسأل: هل هذا اسم شركة، أم نموذج، أم منتج أستخدمه، أم طبقة تشغيل تمنح النموذج أدوات وصلاحيات؟

الخطأ الشائع: مقارنة النموذج بالمنتج

حين يقول شخص إن «Codex أفضل من نموذج آخر»، فقد يكون يقارن منتجاً كاملاً بنموذج وحده. وحين يقول إن «النموذج نفسه أضعف في الموقع»، فقد يكون الفرق الحقيقي في المعلومات التي وصلته، أو الأدوات التي يملكها، أو الصلاحيات الممنوحة له، لا في النموذج وحده.

لذلك، قبل أي مقارنة، اسأل:

- هل أقارن نموذجاً بنموذج؟
- أم منتجاً بمنتج؟
- أم بيئة تشغيل ببيئة تشغيل؟
- وهل يعمل الطرفان بالنموذج نفسه فعلاً، أم أن الاسم فقط متشابه؟

هذا التفريق البسيط يحميك من جزء كبير من ضوضاء المقارنات على الإنترنت.

خلاصة الفصل

الشركة ليست النموذج، والنموذج ليس المنتج، والمنتج ليس الأداة، والوكيل ليس اسماً جديداً للنموذج. افصل الطبقات أولاً، ثم قارن.

2 النموذج: العقل الذي يعمل بالكلام

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
النموذج	Model	أين يعيش الذكاء نفسه؟
حدود المعرفة	Knowledge Cutoff	لماذا لا يعرف النموذج كل ما حدث حتى اليوم؟
الهلوسة	Hallucination	لماذا قد يعطي جواباً مقنعاً لكنه خاطئ؟

تحت كل أداة تستخدمها يوجد **نموذج**. هذا هو الجزء الذي يفهم كلامك، ويكتب، ويشرح، ويلخّص، ويقارن، ويحلل، ويقترح الخطوة التالية.

يمكنك أن تتخيله كعقل شديد القوة يعمل بالكلام. دُرّب على كمية هائلة من النصوص والبيانات، منها مواد متاحة على الإنترنت، فصار قادراً على التعامل مع أسئلة ومهمات لم يرها بهذه الصيغة من قبل. ولا تحتاج في هذا الدليل إلى معرفة كيف جرى التدريب أو كم كلف؛ يكفي أن تعرف أن التدريب ينتهي، ثم يصبح لدينا نموذج جاهز للاستخدام.

النموذج وحده — ماذا يفعل وماذا لا يستطيع؟

يستطيع وحده

- ✓ يفهم كلامك ويحلله
- ✓ يكتب ويشرح ويلخّص
- ✓ يقارن ويقترح الخطوة التالية

لا يستطيع وحده

- ✗ متابعة العالم بعد التدريب
- ✗ معرفة ملفاتك وقراراتك
- ✗ تنفيذ إجراء خارجي حقيقي
- ✗ ضمان صحة كل جواب

معرفته تتوقف عند حد معين

حين ينتهي تدريب النموذج، لا يواصل متابعة العالم وحده. توجد نقطة زمنية تقف عندها معرفته التدريبية، وتسمى **حدود المعرفة Knowledge Cutoff**.

قد تحدّث الشركة النموذج لاحقاً، وقد تعطيه أداة بحث يصل بها إلى معلومات اليوم، لكن هذين أمران مختلفان:

- ما تعلّمه النموذج أثناء التدريب؛
- وما يستطيع الوصول إليه الآن باستخدام أداة.

لذلك قد يعرف تاريخ موضوع كامل، ثم يحتاج إلى البحث كي يعرف خبراً صدر هذا الصباح.

لا يعرف عالمك أنت

النموذج قد يعرف الكثير عن التاريخ والعلوم والكتابة والبرمجة، لكنه لا يعرف تلقائياً من أنت، وما الذي تعمل عليه، وما الملفات الموجودة عندك، وما القرارات التي اتخذتها، وما النتيجة التي تعتبرها جيدة.

كل هذه المعلومات يجب أن تصل إليه أثناء العمل. وسنفهم بعد قليل أين توضع، وكم يستطيع أن يحمل منها.

لا يستطيع التنفيذ وحده

يستطيع النموذج أن يكتب لك رسالة بريد، لكنه لا يستطيع إرسالها وحده. يستطيع أن يكتب كوداً، لكن ظهور الكود في الجواب لا يعني أن البرنامج اشتغل. ويستطيع أن يقترح إنشاء ملف، لكن الملف لن يظهر على جهازك لمجرد أنه اقترحه.

النموذج وحده **عقل يعمل بالكلام**. لكي يبحث في الإنترنت، أو يقرأ ملفاتك، أو يشغّل برنامجاً، أو يرسل بريداً، يحتاج إلى أدوات وإلى طبقة تشغيلية تمنحه القدرة على استخدامها.

وقد يجيب حتى عندما لا يعرف

لأن النموذج يولّد الجواب المتوقع، فقد يملأ أحياناً فجوة في معرفته بجواب يبدو مرتباً وواثقاً لكنه غير صحيح. هذا ما يسمى **الهلوسة Hallucination**.

لهذا لا تجعل جمال الصياغة دليلاً على صحة المعلومة. عندما تكون الحقيقة مهمة، اطلب مصدراً، أو أعطِ النموذج أداة بحث، أو تحقّق من النتيجة بطريقة أخرى.

خلاصة الفصل

النموذج هو العقل: يفهم ويولّد ويقترح، لكن معرفته ليست حية دائماً، ولا يعرف عالمك الخاص، ولا يستطيع أن ينفذ إجراءً خارجياً من دون أدوات.

3 التوكن ونافذة السياق: كيف يدخل الكلام، وكم يتسع النموذج؟

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
الطلب	Prompt	ما الكلام الذي تعطيه أنت للنموذج؟
التوكن	Token	كيف يُقسّم الكلام ويُقاس؟
السياق	Context	ما المعلومات الموجودة أمام النموذج الآن؟
نافذة السياق	Context Window	ما الحد الأقصى لما يستطيع النموذج الرجوع إليه؟

رحلة رسالتك إلى النموذج

أنت تكتب للنموذج كلاماً. هذا الكلام يسمى **الطلب Prompt**: السؤال أو التعليمات التي تريد منه تنفيذها.

لكن النموذج لا يستقبل الجملة بالشكل الذي تراها به على الشاشة. قبل أن تصل إليه، تُقسَّم إلى قطع صغيرة تسمى **Tokens**. قد يكون التوكن كلمة كاملة، أو جزءاً من كلمة، أو علامة ترقيم. ولكل توكن تمثيل رقمي يستطيع النظام معالجته.

ثم تحدث الرحلة التالية:

تكتب طلباً

← يُقسَّم الكلام إلى Tokens

← تصل التوكنات إلى النموذج

← يعالجها ويولّد Tokens جديدة

← يحولها المنتج إلى جواب مقروء على الشاشة

لا تحتاج إلى معرفة الرياضيات الواقعة في المنتصف. المهم أن تعرف أن **التوكن هو الوحدة التي نقيس بها مقدار ما يدخل إلى النموذج وما يخرج منه**.

ولهذا يُستخدم التوكن في قياس ثلاثة أشياء عملية:

1. **السعة:** كم يستطيع النموذج أن يقرأ ويولّد في المرة الواحدة.
2. **التكلفة:** كثير من خدمات النماذج تسعّر ما يقرأه النموذج وما يكتبه بعدد التوكنز.
3. **الحجم:** ملف طويل أو محادثة طويلة يستهلكان توكنز أكثر من طلب قصير.

التوكن لا يساوي كلمة دائماً، ولا يوجد تحويل ثابت صالح لكل اللغات. كتقريب شائع للإنجليزية، ألف توكن تساوي نحو 750 كلمة؛ أما العربية فتختلف طريقة تقسيمها من نموذج إلى آخر، لذلك تعامل مع الرقم بوصفه تقريباً لا مسطرة دقيقة.[9]

ما السياق؟

الطلب الذي تكتبه ليس الشيء الوحيد الذي يصل إلى النموذج. قد تصل معه الرسائل السابقة، وملف رفعته، وتعليمات الأداة، ونتيجة بحث، ومعلومات أخرى أضافها المنتج.

كل ما هو موجود أمام النموذج في هذه اللحظة يسمى **السياق Context**.

إذن:

- **الـPrompt** هو ما تطلبه أنت الآن.
- **الـContext** هو كل ما يراه النموذج وهو ينفذ ذلك الطلب.

قد تكتب كلمة واحدة، بينما يصل إلى النموذج معها آلاف التوكنز من التعليمات والتاريخ والملفات. سنرى كيف يحدث ذلك بالتفصيل في فصل «ماذا يجد النموذج أمامه قبل أن تكتب أول كلمة؟».

قد تكتب كلمة واحدة، بينما يصل معها آلاف التوكنز.

أول حد حقيقي: نافذة السياق

كل نموذج محدود بكمية قصوى يستطيع الرجوع إليها في المرة الواحدة. هذه الكمية تسمى **نافذة السياق Context Window**، وتُقاس بالتوكنز.[2]

تدخل في هذه النافذة أشياء مثل:

- تعليمات النظام؛
- طلبك الحالي؛
- الرسائل السابقة التي أُعيد إرسالها؛
- محتوى الملفات الذي قُرئ؛
- تعريفات الأدوات ونتائجها؛
- والمساحة التي يحتاجها الجواب الجديد.

لذلك كلمة «نافذة» مهمة: النموذج لا يرى كل ما تعرفه الأداة أو كل ما يوجد على جهازك؛ يرى فقط ما أُدخل إلى هذه النافذة الآن.

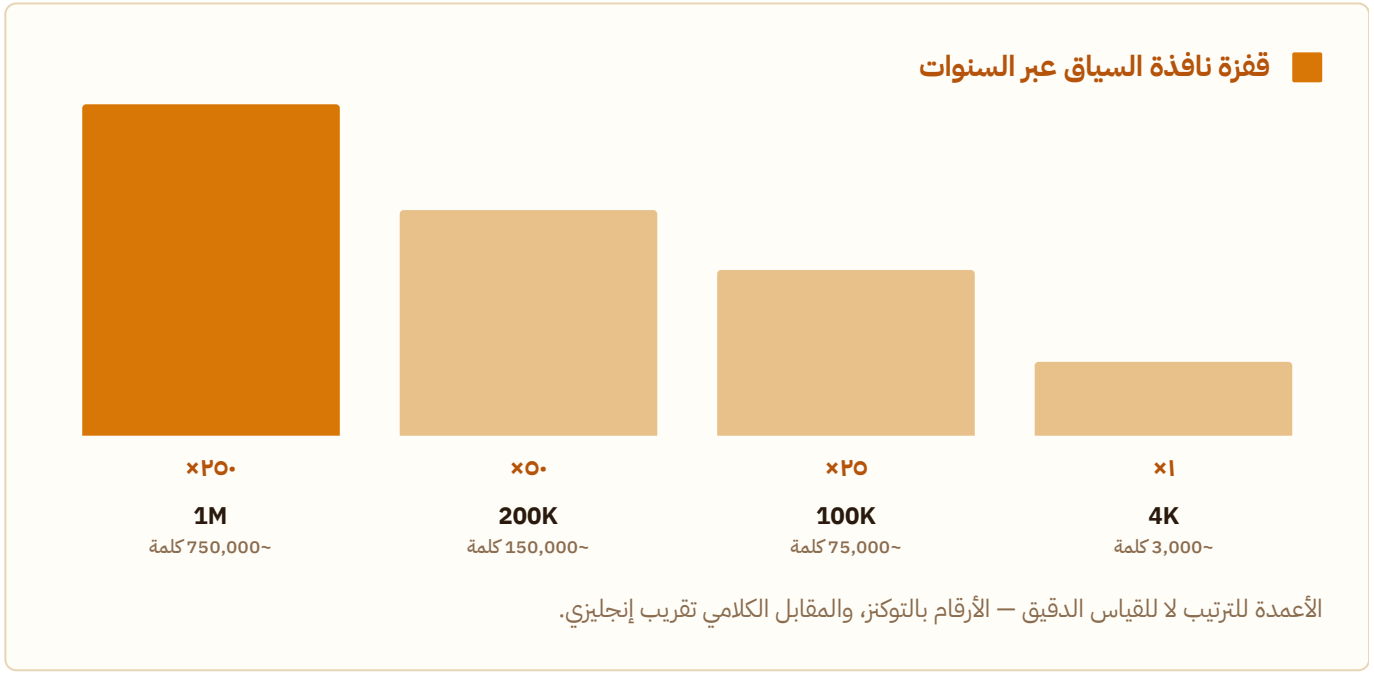
كيف تطورت أحجام النافذة؟

في بدايات هذه الموجة، كانت نماذج شائعة تملك نافذة تقارب **4 آلاف توكن**. وفي مايو 2023 أعلنت Anthropic نافذة **100 ألف**، ثم أعلنت **200 ألف** في نوفمبر من العام نفسه. وفي فبراير 2024 أعلنت Google نموذجاً يصل إلى **مليون توكن**. واليوم أصبحت نافذة المليون متاحة في أكثر من عائلة من النماذج.[10][11][12]

لكي تتخيل الفرق، وباستخدام التقريب الإنجليزي فقط:

حجم النافذة	ما يقابله تقريباً بالإنجليزية	مقارنة بـ4 آلاف
4,000 توكن	3,000 كلمة تقريباً	×1
100,000 توكن	75,000 كلمة تقريباً	×25
200,000 توكن	150,000 كلمة تقريباً	×50
1,000,000 توكن	750,000 كلمة تقريباً	×250

هذه ليست وعوداً بأن النموذج سيتذكر كل تفصيل بالجودة نفسها، وليست أرقاماً ثابتة لكل منتج. النافذة تختلف بحسب النموذج والخطة والأداة، لكنها توضح حجم القفزة: ما كان يتسع لمحادثة قصيرة صار يستطيع استقبال مشروع أو مجموعة كبيرة من الوثائق.



ماذا يحدث عندما تقترب النافذة من الامتلاء؟

لا يعني ذلك بالضرورة أن التطبيق سيغلق المحادثة فجأة. كل منتج يدير الحد بطريقته. قد يفعل واحداً أو أكثر من الآتي:

- يحذف أقدم أجزاء المحادثة من الطلب الجديد؛
- يلخّص ما سبق ويحمل الملخص بدلاً من كل الرسائل؛

- يضغط نتائج الأدوات أو يزيل القديم منها؛
- يرفض ملفاً أو طلباً أكبر من المساحة المتبقية؛
- أو يطلب منك بدء جلسة جديدة.

النتيجة التي تلاحظها أنت قد تكون نسيان تفصيل قديم، أو خلطاً بين قرارات، أو تراجعاً في التركيز. وعندما يحدث ذلك، ابدأ جلسة جديدة مع خلاصة واضحة للحالة بدلاً من الاستمرار في محادثة مزدحمة.

السياق ليس ذاكرة دائمة

قد يحفظ المنتج محادثتك، لكن حفظها في واجهة التطبيق لا يعني أن النموذج يرى كل حرف منها إلى الأبد. لكي يستخدم النموذج معلومة قديمة، يجب أن يعيد المنتج إدخالها أو إدخال ملخص منها إلى السياق الحالي.

إذن نافذة السياق هي **ذاكرة العمل الحالية**، وليست ذاكرة دائمة. أما الذاكرة الدائمة، إن وجدت، فتعيش خارج النموذج ثم تُسترجع منها المعلومات المناسبة عند الحاجة.

ثلاثة أمراض للسياق

- **نقص السياق:** تطلب نتيجة دقيقة من دون إعطاء النموذج المعلومات اللازمة.
- **تلوث السياق:** تملأ الجلسة بمواد لا تحتاجها المهمة، فتدفن المهم وسط الضجيج.
- **تقادم السياق:** تبقى قرارات قديمة أمام النموذج فيتعامل معها كأنها ما زالت صحيحة.

العلاج واحد في جوهره: أعطه المعلومات المرتبطة بالمهمة، وحدّث مصدر الحقيقة، وابدأ سياقاً جديداً عندما يصبح القديم عبئاً.

خلاصة الفصل

كلامك يدخل إلى النموذج على شكل توكنز. وكل ما يراه النموذج يعيش داخل نافذة سياق محدودة. كبرت هذه النافذة كثيراً عبر السنوات، لكنها ما زالت محدودة، وما يوجد خارجها غير موجود أمام النموذج الآن.

4 الأدوات: كيف يتحول الكلام إلى فعل؟

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
الأداة	Tool	ما القدرة الإضافية التي يمكن أن نمنحها للنموذج؟
استدعاء الأداة	Tool Calling	كيف يطلب النموذج استخدام القدرة؟
الصلاحية	Permission	ما الذي نسمح للأداة أن تفعله فعلياً؟

وصلنا الآن إلى لحظة التحول الكبرى.

لدينا نموذج يستطيع أن يفهم طلبك، ويحلله، ويقترح خطة لتنفيذه. لكنه، إلى هذه اللحظة، ما زال قادراً على **الحديث عن العمل** أكثر من قدرته على **تنفيذ العمل**.

تستطيع أن تقول له:

> **ابحث في أحدث المعلومات عن هذا الموضوع، وقارن المصادر، واكتب تقريراً، واحفظه، ثم أرسله إلى بريدي.**

إن لم تكن حوله أدوات، فقد يكتب لك تقريراً اعتماداً على معرفته الحالية، أو يخبرك بخطوات البحث، أو يصوغ رسالة البريد. لكنه لا يستطيع وحده أن يفتح محرك بحث، أو يكتب ملفاً على جهازك، أو يرسل رسالة حقيقية.

هنا تظهر **الأداة Tool**.

ما الأداة؟

الأداة هي قدرة تنفيذية نعطيها للنظام ليجري عملاً خارج الكلام. لكل أداة اسم، ووصف لما تفعله، ومعلومات تحتاجها كي تعمل، ونتيجة تعيدها بعد التنفيذ.

أمثلة:

الأداة	ماذا نعطيها؟	ماذا تعيد؟
البحث في الويب	ما الذي نبحث عنه؟	نتائج البحث
قراءة ملف	أي ملف؟	محتوى الملف
كتابة ملف	أين نحفظه، وماذا نكتب؟	تأكيد الحفظ أو رسالة خطأ
إنشاء صورة	وصف الصورة	الصورة الناتجة
تشغيل برنامج	اسم البرنامج وما المطلوب منه	النتيجة أو الخطأ الذي ظهر
إرسال بريد	المستلم والعنوان والنص والمرفق	تأكيد الإرسال أو فشله

لكن السؤال الذي يربك كثيراً من الناس هو: **كيف يستخدم النموذج أداة وهو لا يملك يداً تضغط عليها؟**

النموذج لا ينفذ الأداة بنفسه

الإجابة بسيطة: النموذج **يطلب** استخدام الأداة، والبرنامج المحيط به هو الذي **يشغّلها فعلياً**.

تحدث العملية في خمس خطوات:

1. تخبر الطبقة التشغيلية النموذج بالأدوات الموجودة لديه.
2. يختار النموذج الأداة المناسبة ويكتب لها ما يريده.
3. تشغّل الطبقة التشغيلية الأداة فعلياً.
4. تعود نتيجة الأداة إلى النموذج.
5. يقرأ النموذج النتيجة ويكمل العمل.

تسمي وثائق المنصات هذه الآلية **Tool use** أو **Tool calling**. وتوضح صراحة أن النموذج لا ينفذ العملية الخارجية بذاته؛ بل يُصدر طلباً منظماً، ثم تنفذه جهة أخرى وتعيد إليه النتيجة.[3]

مثال: من الطلب إلى تقرير مُرسَل

لنفترض أنك قلت للوكيل:

> **ابحث في أحدث المعلومات عن موضوع معين، وقارن المصادر، واكتب تقريراً، واحفظه، ثم أرسله إلى بريدي.**

هذا طلب واحد بالنسبة إليك، لكنه يحتاج إلى عدة أدوات وراء الشاشة.

الخطوة الأولى: البحث

يلاحظ النموذج أن الطلب يتحدث عن «معلومات حديثة»، فيقرر استخدام أداة البحث. يولّد طلباً مثل:

«ابحث عن أحدث المعلومات الموثوقة عن هذا الموضوع». تنفذ بيئة التشغيل البحث، ثم تعيد النتائج إلى السياق.

الخطوة الثانية: القراءة والمقارنة

يقرأ النموذج النتائج، وقد يقرر فتح عدد من المصادر. يستدعي أداة جلب الصفحات أو القراءة. تعود النصوص إليه، فيقارن التواريخ والادعاءات والأدلة.

الخطوة الثالثة: إنشاء التقرير

حين تتوفر لديه المادة، يكتب التقرير، ثم يطلب من أداة الملفات إنشاء ملف حقيقي في المكان الذي حددته.

الخطوة الرابعة: الإرسال

بعد نجاح الحفظ، يطلب استخدام أداة البريد بالمستلم والعنوان والنص والمرفق. وقد تتوقف الطبقة التشغيلية وتطلب موافقتك قبل الإرسال، لأنه إجراء يؤثر في العالم خارج المحادثة.

لاحظ ما حدث: لم تُنفَّذ المهمة بضغطة سحرية واحدة. النموذج اتخذ سلسلة قرارات، والأدوات نفذت الأفعال، والنتائج عادت إلى النموذج ليقرر ما بعدها.

الأداة لا تعطي النموذج معرفة جديدة فقط، بل قدرة جديدة

حين تعطي النموذج أداة بحث، لا تكون قد درّبته من جديد، بل جعلت نظامه قادراً على الوصول إلى معلومات حية.
وحين تعطيه أداة كتابة، لا تجعله أفضل أدباً، بل تمنحه طريقة لتحويل النص إلى ملف حقيقي. وحين تعطيه أداة بريد، لا تزيد ذكاءه، بل توسّع نطاق أفعاله.

لهذا من المفيد أن تفرّق بين سؤالين:

- **ما الذي يستطيع النموذج فهمه أو توليده؟**
- **ما الذي يستطيع النظام تنفيذه في العالم؟**

الأول يتعلق بقدرة النموذج. الثاني يتعلق بالأدوات والوصول والصلاحيات.

الأداة والصلاحية شيئان مختلفان

قد يمتلك النظام أداة بريد، لكن يُسمح له بالقراءة فقط. وقد يمتلك أداة ملفات، لكن داخل مجلد واحد. وقد يعرف كيف يحذف ملفاً، لكن البيئة تطلب موافقتك قبل الحذف.

الأداة تقول: **هذه العملية ممكنة**.

الصلاحية تقول: **متى، وعلى ماذا، وبأي حدود، يمكن تنفيذها**.

قدرة الأداة لا تعني أن كل أبوابها مفتوحة. النظام الجيد يمنح الوكيل أقل صلاحية تكفي لإنجاز العمل، ويطلب موافقتك قبل الإجراءات الحساسة مثل الحذف أو الإرسال أو النشر.

خلاصة الفصل

الأداة تمنح النموذج إجراءً يستطيع طلبه. النموذج يطلب، والطبقة التشغيلية تنفذ، ثم تعيد إليه النتيجة. بهذه الآلية ينتقل الذكاء الاصطناعي من الكلام عن العمل إلى تنفيذ العمل.

5 الطبقة التشغيلية: لماذا يبدو النموذج مختلفاً من مكان إلى آخر؟

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
الطبقة التشغيلية	Harness	ما البرنامج الذي يحيط بالنموذج ويحدد طريقة عمله؟
بيئة العمل	Environment	أين توجد الملفات والبرامج والخدمات التي يستطيع استخدامها؟

الآن لدينا نموذج، وسياق، وأدوات. لكننا نحتاج إلى برنامج يجمع هذه القطع ويجعلها تعمل معاً. هذه هي **الطبقة التشغيلية**، ويُشار إليها بالإنجليزية عادةً باسم **Harness**.

ما الطبقة التشغيلية؟

هي البرنامج المحيط بالنموذج، والذي يحوّله من عقل يكتب النص إلى نظام يستطيع العمل. وهي:

- ترسل كلامك إلى النموذج وتعرض لك جوابه؛
- تعطيه تعليماته الأساسية؛
- تخبره بالأدوات المتاحة؛
- تشغّل الأدوات عندما يطلبها؛
- تعيد نتائج الأدوات إليه؛
- تدير نافذة السياق والمحادثة؛
- وتحدد الملفات والخدمات والصلاحيات المتاحة.

أما **بيئة العمل Environment** فهي المكان الفعلي الذي توجد فيه الأشياء التي سيعمل عليها: متصفح، أو مجلد على كمبيوترك، أو خادم، أو قاعدة بيانات، أو خدمة متصلة.

الفكرة الأهم: **النموذج لا يعيش وحده؛ هناك نظام كامل حوله يحدد ما يراه وما يستطيع فعله.**

المنتج الذي تستخدمه ليس مجرد نافذة للنموذج

عندما تفتح منتجاً مثل واجهة محادثة أو بيئة عمل على الكمبيوتر، فأنت لا تصل إلى النموذج في فراغ. أنت تصل إلى **منتج كامل حول النموذج**.

هذا المنتج يحدد أشياء مثل:

- النموذج أو مجموعة النماذج المستخدمة؛
- تعليمات لا تراها؛
- أدوات مبنية مسبقاً؛
- طريقة لإدارة المحادثة والسياق؛
- اتصالات بخدمات خارجية؛
- نظام ملفات أو مساحة عمل؛
- قواعد أمان وموافقات؛
- طريقة لحفظ الجلسات أو استئنافها؛
- وواجهة تفاعلك معه.

ولهذا قد يبدو النموذج أكثر قدرة في مكان وأقل قدرة في مكان آخر، حتى عندما تختار اسم النموذج نفسه.

نفس الدماغ، وظيفتان مختلفتان

لنقارن طريقتين يمكن أن يعمل فيهما نموذج قوي:

محادثة في المتصفح	وكيل يعمل على مشروعك أو كمبيوترك
أمثلة: ChatGPT، وClaude، وGemini	أمثلة: Claude Code، وCodex، وGoogle Antigravity، وOpenCode[13]
يبدأ مما تكتبه وما ترفعه وما توفره المنصة	يبدأ داخل مجلد أو مشروع فعلي
قد يبحث، ويحلل ملفات، ويكتب لك كوداً	يستطيع قراءة ملفات المشروع وتعديلها
قد يعطيك الكود لتنسخه أو يعرض نتيجة داخل المنتج	يستطيع تشغيل البرنامج والاختبارات ورؤية الأخطاء
يبقى العمل غالباً داخل حدود المحادثة والخدمات المدمجة	يحفظ تغييرات حقيقية، ثم يعود ليفحصها ويصلحها

إذا طلبت من محادثة عادية: «ابنِ لي هذا التطبيق»، فقد تكتب لك الملفات أو الكود وتشرح ما يجب فعله. أما داخل Claude Code أو Codex، فيستطيع الوكيل فتح المشروع نفسه، وإنشاء الملفات، وتشغيل التطبيق، ورؤية رسالة الخطأ، ثم تعديل ما كتبه.

العقل لم يتعلم البرمجة من جديد. الذي تغيّر هو العالم الذي يراه، والأدوات التي يملكها، وقدرته على رؤية نتيجة التنفيذ.

هذه هي الفكرة المقصودة عندما نقارن Claude مع Claude Code، أو ChatGPT مع Codex. قد تختلف نسخة النموذج بين منتج وآخر، لكن قوة النموذج وحدها لا تشرح فرق القدرة بينهما؛ البيئة والأدوات تصنعان فرقاً أساسياً.

ثلاثة أنماط لاستخدام الذكاء الاصطناعي

1. محادثة في المتصفح

تدخل إلى ChatGPT أو Claude أو Gemini، وتبدأ الحديث. هذا هو الأسهل للطلبات السريعة: سؤال، وصياغة، وتلخيص، وبحث، وتحليل ملف ترفعه، وإنشاء مخرج داخل المنتج.

في هذا النمط، أنت تعمل داخل الأدوات والحدود التي اختارتها المنصة، ولا يعيش الوكيل تلقائياً داخل ملفات كمبيوترك ومشروعاتك.

2. وكيل يعمل على مشروع أو كمبيوتر

تفتح Claude Code أو Codex أو Antigravity أو OpenCode داخل مساحة عمل. هنا يستطيع الوكيل، ضمن الصلاحيات التي تمنحها، أن يقرأ الملفات ويعدّلها، ويشغّل البرامج، ويرى النتائج، ويواصل العمل.

هذا النمط هو الأنسب عندما تريد أن **يعمل الوكيل على الشيء نفسه**، لا أن يكتفي بإعطائك كلاماً عنه: بناء تطبيق، مراجعة مجلد وثائق، تجهيز تقرير داخل مشروع، أو تشغيل عملية على ملفاتك.

3. طبقة تشغيلية تبنيها أنت أو شركة

قد تريد نظاماً لا يشبه أداة عامة، بل يؤدي وظيفة محددة: يستقبل طلبات العملاء، أو يراجع الفواتير، أو يعد تقريراً أسبوعياً، أو يبحث في معرفة الشركة.

هنا تُبنى بيئة خاصة لهذه المهمة، وتُعطى المعلومات والأدوات والصلاحيات التي تحتاج إليها فقط.

من أين تأتي القوة العملية؟

النموذج جزء واحد فقط من قوة النظام. عندما تريد فهم ما يستطيع منتج فعله فعلياً، انظر إلى ثلاثة أمور:

1. **قوة النموذج:** هل يستطيع فهم المهمة والتخطيط لها جيداً؟
2. **ما يعرفه عن المهمة:** هل يرى ملفات المشروع وقواعدك والقرارات السابقة، أم يبدأ من رسالة قصيرة فقط؟
3. **ما يستطيع فعله:** هل يجيب بالكلام فقط، أم يستطيع البحث والكتابة والتشغيل والوصول إلى الخدمات المطلوبة؟

مثال: نموذج قوي داخل محادثة لا تعرف شيئاً عن شركتك سيبدأ من الصفر. وإذا وضعت النموذج نفسه داخل مشروع فيه ملفاتك وتعليماتك، ومنحته أدوات القراءة والكتابة، ستتغير النتيجة جذرياً. وإذا أوصلته أيضاً بقاعدة بيانات أو بريد، صار نطاق عمله أوسع مرة أخرى.

القوة العملية تظهر حين تجتمع جودة العقل مع المعلومات الصحيحة والقدرة على الفعل.

خلاصة الفصل

النموذج ليس المنتج الكامل. الطبقة التشغيلية تحدد ما يراه، وما الأدوات التي يملكها، وأين يستطيع العمل. ولهذا يمكن للعقل نفسه أن يبدو كاتباً داخل محادثة، وعاملاً ينفذ ويختبر داخل مشروع على الكمبيوتر.

6 الوكيل: حين يبدأ النظام بالعمل في حلقة

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
الوكيل	Agent	متى يصبح النموذج نظاماً يعمل نحو هدف؟
الحلقة الوكيلية	Agentic Loop	كيف يختار الوكيل الخطوة التالية؟
سير العمل	Workflow	ما الفرق بين خطوات ثابتة ومسار يقرره الوكيل؟
الاستقلالية	Autonomy	ما مقدار العمل الذي نسمح له بتنفيذه وحده؟

تُستخدم كلمة **Agent** اليوم بكثرة حتى كادت تفقد معناها. يضعها بعض المنتجات على أي محادثة، ويستخدمها آخرون لنظام يعمل ساعات وينفذ عشرات الخطوات.

لنأخذ أبسط تعريف مفيد:

> **حين تعطي النظام هدفاً وأدوات، ويصبح النموذج قادراً على اختيار أداة، ورؤية نتيجتها، ثم اختيار الخطوة التالية حتى ينجز المهمة، يصبح لدينا وكيل.**

تعرّف Anthropic الوكيل بصورة قريبة: نموذج يوجّه عملياته واستخدام أدواته، ويعمل في حلقة يخطط فيها، ويفعل، ويراقب النتيجة، ثم يعدّل حتى ينتهي أو يحتاج إلى تدخل الإنسان.[4]

الوكيل ليس نموذجاً جديداً

حين تحول النموذج إلى وكيل، لم تنشئ نوعاً آخر من الذكاء داخل مركز البيانات. أنت وضعت النموذج داخل نظام يملك:

- هدفاً؛
- سياقاً؛

- أدوات؛
- صلاحيات؛
- قواعد؛
- وحلقة تسمح له باختيار الخطوة التالية بناءً على النتيجة السابقة.

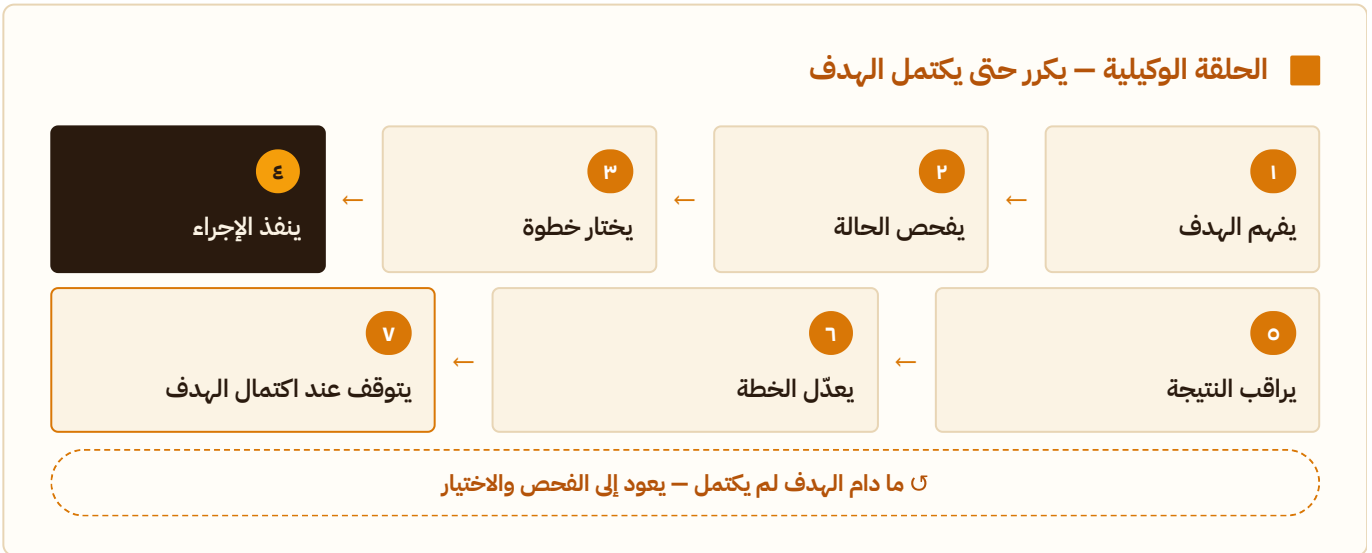
لذلك يمكن القول:

> **وكيل عملي = نموذج + هدف + سياق + أدوات + حلقة عمل + حدود**

الحلقة الوكيلية Agentic Loop

تأخذ الحلقة شكلاً قريباً من هذا:

1. **يفهم الهدف.** ماذا يريد المستخدم في النهاية؟
2. **يفحص الحالة الحالية.** ما المعلومات والملفات والنتائج الموجودة؟
3. **يختار خطوة.** هل يجيب، أم يبحث، أم يقرأ، أم يكتب، أم يشغّل أداة؟
4. **ينفذ الإجراء.** عبر الأداة والطبقة التشغيلية.
5. **يراقب النتيجة.** هل نجحت؟ هل ظهرت معلومة أو مشكلة جديدة؟
6. **يعدّل الخطة.** يكرر أو يصلح أو يطلب توضيحاً.
7. **يتوقف.** عندما يكتمل الهدف، أو يعجز، أو يصل إلى قرار يحتاج الإنسان.



كيف تبدو الحلقة في مثال التقرير؟

نطلب من الوكيل: «ابحث، وقارن، واكتب، واحفظ، وأرسل».

قد يعمل هكذا:

يفهم المطلوب

→ يبحث عن مصادر حديثة

→ يقرأ النتائج

→ يلاحظ أن مصدراً أساسياً محجوب

→ يبحث عن مصدر بديل

→ يقارن الادعاءات

→ يكتب مسودة

→ يحفظ الملف

→ يفتح الملف ليتأكد من شكله

→ يطلب موافقتك على الإرسال

→ يرسل البريد

→ يبلغك بما أنجزه وما تعذر عليه

لاحظ أن ترتيب الخطوات لم يُكتب كله مسبقاً كقائمة جامدة. الوكيل اختار بعض الخطوات استجابةً لما حدث أثناء التنفيذ.

هذه المرونة هي جوهر عمل الوكيل، لكنها أيضاً مصدر الخطر.

Workflow أم Agent؟

ليس كل نظام يستخدم نموذجاً نظاماً وكيلياً؛ فقد يدخل النموذج في سير عمل ثابت من دون أن يختار مساره بنفسه.

سير عمل ثابت Workflow

تحدد الخطوات مسبقاً:

استقبل قائمة المصادر المحددة

→ اجلب محتواها

→ ضع الملخصات في قالب التقرير

→ احفظ الناتج في المكان المتفق عليه

المسار معروف، ولا يملك النموذج حرية كبيرة في تغييره.

وكيل Agent

تعطيه هدفاً وأدوات وحدوداً، ويقرر المسار داخلها:

ابحث في هذا الموضوع الحديث، واختر أفضل المصادر، وافحص التعارضات، ثم أنشئ التقرير وارجع إليّ قبل إرسال أي شيء.

قد يقرر قراءة ملف إضافي، أو طلب توضيح، أو مراجعة تعارض، أو تقسيم العمل.

لا يوجد فائز مطلق بين الاثنين. إن كانت العملية معروفة وثابتة، فقد يكون سير العمل المحدد أبسط وأكثر موثوقية. وإن كانت المهمة تتطلب حكماً وتكيفاً مع نتائج متغيرة، قد يكون الوكيل أنسب.

الاستقلالية ليست مفتاح تشغيل وإيقاف

يمكن أن يعمل الوكيل على درجات:

1. **يقترح فقط:** يعرض الخطة ولا ينفذ.
2. **ينفذ بعد كل موافقة:** يطلب إذناً لكل فعل مهم.
3. **ينفذ داخل حدود:** يعمل تلقائياً في مجلد أو نظام محدد.
4. **يتوقف عند نقاط حساسة:** مثل الإرسال، والنشر، والشراء، والحذف.
5. **يعمل باستقلالية أوسع:** ضمن مهمة طويلة مع مراقبة وسجلات وحدود واضحة.

التصميم الجيد لا يسأل فقط: «ما الذي يستطيع الوكيل فعله؟» بل أيضاً: **ما الذي ينبغي أن نسمح له بفعله وحده؟**

لماذا قد يخطئ الوكيل رغم قوة النموذج؟

لأن الخطأ يمكن أن يأتي من كل طبقة:

- فهم الهدف بصورة ناقصة؛
- سياق قديم أو ملوث؛
- معلومة غير صحيحة من مصدر خارجي؛
- اختيار أداة غير مناسبة؛
- مدخلات خاطئة للأداة؛
- صلاحية أوسع من المطلوب؛
- نتيجة لم تُفحص؛
- تعليمات خبيثة داخل صفحة أو ملف يحاول الوكيل قراءته؛
- أو معيار توقف غير واضح.

لهذا قوة الوكلاء لا تلغي دور الإنسان. كلما صار الفعل أكثر تأثيراً، احتجت إلى:

- حدود وصلاحيات حقيقية؛
- موافقات عند النقاط الحساسة؛
- سجلات لما فُعل؛
- اختبار للنتائج؛
- ومكان واضح يستطيع فيه الوكيل التوقف وطلب المساعدة.

خلاصة الفصل

الوكيل ليس اسماً آخر للنموذج، بل طريقة تشغيله داخل هدف وأدوات وحدود وحلقة يكرر فيها خطواته بناءً على النتائج. المرونة التي تجعله قوياً هي نفسها التي تجعل الاختبار والصلاحيات والإشراف ضرورية.

7 ماذا يجد النموذج أمامه قبل أن تكتب أول كلمة؟

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
تعليمات النظام	System Prompt / System Instructions	من يخبر النموذج بدوره وقواعده؟
تعريفات الأدوات	Tool Definitions	كيف يعرف النموذج ما الأدوات الموجودة لديه؟
تعليمات المشروع	Project Instructions	كيف تعطي الوكيل قواعدك أنت؟
الذاكرة	Memory	ما الذي نحفظه ليعود في جلسة لاحقة؟
الحالة	State	أين وصل العمل، وما الخطوة التالية؟

افتح جلسة جديدة في إحدى أدوات الوكلاء، ثم انظر إلى نافذة السياق إن كانت الأداة تعرضها. قد تجد أن جزءاً منها مستخدم قبل أن تكتب كلمة واحدة.

لماذا؟

لأن رسالتك ليست الشيء الوحيد الذي يصل إلى النموذج.

لكي يعمل النموذج داخل منتج أو مشروع، يحتاج إلى **إحاطة أولية**. الطبقة التشغيلية تجمع هذه الإحاطة وترسل المناسب منها إلى النموذج.

إذا كنت تستخدم Claude Code، يمكنك كتابة `/context` لترى كيف تتوزع نافذة السياق بين التعليمات والأدوات والمحادثة والملفات وغيرها. لن يعرض لك بالضرورة النص الداخلي الكامل، لكنه يريك أن الجلسة لم تبدأ من الصفر. كما يعرض `/memory` ملفات التعليمات والذاكرة التي حُمّلت.[14]

نافذة السياق قبل أول كلمة تكتبها

الجلسة لا تبدأ من الصفر — الإحاطة الأولية تستهلك جزءاً من النافذة قبل رسالتك.

1. تعليمات النظام System Instructions

تسمى أيضاً **System Prompt**. وهي القواعد الأساسية التي تضعها الجهة التي بنت الطبقة التشغيلية. تعليمات Claude Code تكتبها Anthropic، وتعليمات Codex تكتبها OpenAI، وإذا بنيت وكيلاً خاصاً بك فستكتب أنت تعليماته الأساسية.

قد تحدد هذه التعليمات:

- دور النموذج؛
- أسلوب استخدام الأدوات؛
- قواعد الأمان؛
- متى يطلب الموافقة؛
- كيف يعرض النتائج؛
- وما الذي لا يجوز له فعله.

غالباً لا يرى المستخدم هذه التعليمات كاملة، لكنها تؤثر في سلوك النظام.

2. تعريفات الأدوات

لكي يختار النموذج أداة، يجب أن يعرف بوجودها. لذلك يتلقى عادةً اسم الأداة ووصفها وشكل المدخلات التي تقبلها.

مثلاً، لا يكفي أن تكون هناك أداة لكتابة الملفات داخل البرنامج. يجب أن يعرف النموذج اسمها، وأنها تكتب ملفاً، وما المعلومات التي يجب أن يرسلها لها.

في الأنظمة الصغيرة قد تصل تعريفات الأدوات كلها. وفي الأنظمة الكبيرة قد يرى النموذج أسماءً وأوصافاً مختصرة أولاً، ثم تُحمّل التفاصيل عند الحاجة. بهذه الطريقة لا تُستهلك نافذة السياق في شرح أدوات لن يستخدمها.

3. تعليمات المستخدم أو المشروع

قد يكون لديك ملف أو مساحة تشرح للنظام:

- من أنت؛
- ما المشروع؛
- ما القواعد التي يجب اتباعها؛
- ما بنية المجلدات؛
- ما المعايير؛
- وما القرارات السابقة.

في Claude Code قد تكون هذه التعليمات داخل `CLAUDE.md` ، وفي Codex قد توجد داخل `AGENTS.md` . وفي منتجات أخرى قد تكون إعدادات مشروع أو تعليمات مخصصة.

الاسم والآلية يتغيران. المبدأ ثابت: **أنت تحفظ معلومات دائمة خارج النموذج، ثم تجعل البيئة تعيد المناسب منها إليه عند العمل.**

4. تاريخ المحادثة

لكي يفهم النموذج أن رسالتك الحالية متابعة لما سبق، ترسل البيئة جزءاً من تاريخ الجلسة أو ملخصاً له. فإذا كانت المحادثة طويلة، قد لا يعود كل حرف قديم كما هو؛ قد يُختصر أو يُحذف بحسب طريقة إدارة السياق.

5. طلبك الحالي

بعد هذه الإحاطة يأتي طلبك الحالي: الرسالة التي كتبتها، والملفات التي رفعتها معها، وأي تفاصيل أضفتها لهذه المهمة. وهذا هو الجزء الذي توجّه به النموذج الآن، لكنه ليس كل ما يراه.

الملف الموجود ليس بالضرورة في السياق

هذه نقطة دقيقة ومهمة.

قد يمتلك الوكيل **صلاحية الوصول** إلى مجلد فيه ألف ملف، لكن هذا لا يعني أن محتوى الألف ملف موجود أمام النموذج الآن. الوصول يعني أنه يستطيع استخدام أداة للبحث أو القراءة. ولا يدخل المحتوى في السياق إلا عندما تقرأه البيئة أو تسترجع الجزء المناسب منه وتقدمه للنموذج.

إذن فرّق بين:

- **وجود المعلومة في العالم**؛
- **قدرة النظام على الوصول إليها**؛
- **وجودها فعلياً في السياق الحالي**؛
- **وانتباه النموذج إليها واستخدامه الصحيح لها**.

هذه أربع مراحل، وليست شيئاً واحداً.

Prompt وContext وMemory وState

المصطلح	السؤال الذي يجيب عنه
Prompt	ماذا أطلب الآن؟
Context	ما الذي يراه النموذج وهو ينفذ الطلب؟
Context window	ما السعة القصوى لما يستطيع الرجوع إليه الآن؟
Memory	ما المعلومات التي نحفظها لاستعمالها لاحقاً؟
State	أين وصلت المهمة، وما الذي حدث وما الذي بقي؟

الState مهم جداً في المهمات الطويلة. قد يشمل قائمة بما أُنجز، والملفات الناتجة، والأخطاء، والخطوة التالية. إذا كانت الحالة موجودة فقط داخل محادثة تقترب من الامتلاء، فقد تضيع استمرارية العمل. أما حين تُكتب في ملف أو قاعدة بيانات أو سجل واضح، يستطيع وكيل جديد أو جلسة جديدة استئنافها.

ملفاتك هي الذاكرة التي تملكها

محادثتك مع الوكيل قد تختفي، أو يتغير أسلوب تلخيصها، أو تنتقل أنت إلى أداة أخرى. أما الملف الذي تملكه فيمكنك قراءته وتعديله ونقله وإدخاله إلى أكثر من بيئة.

لذلك من أفضل عادات العمل مع الوكلاء:

- اجعل القرارات المهمة مكتوبة؛

- احفظ حالة المشروع؛
- دوّن ما جُرّب وما فشل؛
- ضع القواعد في مصدر واضح؛
- وافصل الحقائق الحالية عن السجل التاريخي؛
- ولا تجعل المحادثة وحدها مصدر الحقيقة.

مثلاً، بدل أن تعيد في كل مرة شرح جمهور التقرير، وشكله، والمصادر المقبولة، ومكان حفظه، يمكنك وضع هذه القواعد في ملف واضح داخل المشروع. حين يقرأه الوكيل، يبدأ من طريقتك الحقيقية بدلاً من التخمين.

خلاصة الفصل

النموذج لا يبدأ من رسالتك وحدها. الطبقة التشغيلية تعطيه إحاطة أولية من التعليمات، وتعريفات الأدوات، وتاريخ المحادثة، وملفات المشروع، وما استُرجع من الذاكرة. ولهذا قد تبدأ جلسة جديدة وجزء من نافذة السياق مستخدم بالفعل.

8 كيف تمنح الوكيل طريقة عملك وقدرات جديدة؟

مصطلحات هذا الفصل

المصطلح	بالإنجليزية	ما الذي ستفهمه؟
المهارة	Skill	كيف تكتب طريقة عملك مرة واحدة ليعيد الوكيل استخدامها؟
واجهة البرمجة	API	كيف تتعامل البرامج مع خدمة خارجية؟
مفتاح الوصول	API Key	كيف تسمح الخدمة لأداتك باستخدامها؟
ملف الأسرار	`.env`	أين تحفظ المفاتيح بعيداً عن المحادثة والكود؟
بروتوكول MCP	Model Context Protocol	كيف تعرض خدمة خارجية أدواتها وبياناتها للوكيل؟
قاعدة البيانات	Database	أين تعيش بيانات العمل المنظمة؟
الوكيل الفرعي	Sub-agent	كيف يفوّض الوكيل مهمة إلى عامل مستقل؟
الأتمتة	Automation	كيف تبدأ العملية وتتكرر من دون إعادة تشغيلها يدوياً؟
التنسيق	Orchestration	كيف ترتب عدة خطوات وأدوات ووكلاء في عملية واحدة؟

لا تحتاج الآن إلى بناء كل هذه الأشياء أو إتقانها. يكفي أن تفهم المشكلة التي يحلها كل واحد منها، كي تعرف ما الذي تحتاجه عندما يتوقف الوكيل أمام حد معين.

أولاً: درّب الوكيل على طريقتك في العمل

Skill: دليل عمل قابل لإعادة الاستخدام

لنفترض أنك تكرر شرح الطريقة نفسها للوكيل: ابدأ بملخص، وافحص تاريخ كل مصدر، وافصل الحقيقة عن الاستنتاج، واستخدم قالباً محدداً، ولا ترسل شيئاً قبل موافقتي.

بدلاً من إعادة هذا الشرح في كل مرة، تكتبه في **مهارة Skill**. المهارة هي طريقة عمل محفوظة يستطيع الوكيل قراءتها عندما يحتاج إلى ذلك النوع من المهمات. وقد تضم تعليمات، وخطوات، ومعايير، وأمثلة، وقوالب، ومراجع، وأحياناً برامج مساعدة.[5]

الفرق بين الأداة والمهارة

> **الأداة تقول للوكيل: هذا إجراء تستطيع تنفيذه.**
>
> **المهارة تقول له: هذه هي الطريقة التي نؤدي بها هذا النوع من العمل.**

أداة البريد تمنحه قدرة الإرسال. مهارة كتابة التقرير قد تقول له متى يرسل، وما الذي يجب فحصه قبل الإرسال، وكيف يصوغ الرسالة.

والSkill لا تخلق قدرة أو صلاحية من العدم. يمكنك أن تكتب فيها «أرسل البريد»، لكن إن لم تكن للوكيل أداة بريد وصلاحية مناسبة فلن يستطيع التنفيذ.

أربع طرق لاستخدام المهارات

هذه ليست أربعة أنواع تقنية مغلقة، بل أربع درجات عملية عرضناها في فيديو الSkillls، من الأبسط إلى الأقوى:

الاستخدام	ماذا تقول للوكيل؟	مثال
مهمة تكررها	«نفّذ هذا العمل كلما طلبته.»	لخّص المحادثة واحفظ الخلاصة، أو نظّف مجلد التنزيلات وفق قواعد محددة.
هوية تمنحها له	«فكّر وتصرّف بهذه الطريقة.»	كن ناقداً شريكاً، أو معلماً صبوراً، أو آلة تسأل أسئلة تكشف النقص.
خبرة تنقلها إليه	«اعمل وفق هذا المنهج والمعيار.»	راجع القوائم المالية بمنهج محاسب، أو راجع عقداً وفق قائمة فحص قانونية.
تنسيق منظومة كاملة	«شغّل هذه العملية بكل أجزائها.»	وزّع البحث على عدة وكلاء، واجمع النتائج، وأنشئ تقريراً نهائياً واحفظه.

الفرق بين **الهوية** و**الخبرة** مهم: الهوية موقف أو طريقة تفكير يتبناها الوكيل، أما الخبرة فهي منهج أو معيار يمكن نقله ومشاركته.

وفي الاستخدام الرابع يظهر **التنسيق Orchestration**. أنت لا تعطي الوكيل مهمة واحدة، بل تصف له كيف تتحرك خطوات وأدوات ووكلاء متعددون حتى تخرج النتيجة النهائية. وقد تحفظ هذا التنسيق داخل Skill كي يستطيع تشغيل المنظومة مرة أخرى.

ثانياً: فتح الوصول والاتصال

API: حين تحتاج إلى قدرة غير موجودة

لنفترض أن وكيلك لا يستطيع تفريغ تسجيل صوتي، لكن توجد خدمة على الإنترنت متخصصة في تحويل الصوت إلى نص. أو أنه لا يستطيع إرسال بريد، لكن توجد خدمة بريد تستطيع فعل ذلك.

كيف يطلب برنامج داخل وكيلك هذه الخدمة؟ من خلال **API، واجهة برمجة التطبيقات**.

يمكنك التفكير في الـAPI بوصفها الباب الذي فتحته الخدمة للبرامج. الموقع والتطبيق هما الباب المخصص للبشر؛ أما الـAPI فهي الطريقة التي يرسل بها برنامج طلباً إلى الخدمة ويستقبل النتيجة.[6]

بهذه الطريقة يمكن لأداة داخل الوكيل أن تطلب من خدمة خارجية:

- تفريغ تسجيل؛
- توليد صورة؛
- جلب بيانات حديثة؛
- إرسال بريد؛
- قراءة موعد أو إضافته؛
- أو تحديث سجل في نظام عملاء.

API Key وملف `.env`

غالباً تعطيك الخدمة رمزاً سرياً يسمى **API Key**. المفتاح ليس هو الـAPI؛ هو السر الذي يثبت للخدمة أن الطلب صادر من حساب مسموح له باستخدامها.

تعامل معه مثل كلمة المرور:

- لا تلصقه داخل المحادثة؛
- لا تكتبه مباشرة داخل الكود؛

- ولا ترفعه إلى GitHub أو ترسله إلى شخص آخر.

في المشاريع يُحفظ هذا النوع من الأسرار عادةً في ملف اسمه `.env`. الأداة تقرأ المفتاح من الملف عند التنفيذ، وترسله إلى الخدمة من الخلفية، من دون أن يحتاج النموذج إلى رؤية قيمته داخل نافذة السياق.

معرفة النموذج باسم الخدمة لا تكفي. يجب أن توجد أداة أو وصلة حقيقية تستخدم الـAPI والمفتاح لتنفيذ الطلب.

MCP: معيار لوصل تطبيقات الذكاء الاصطناعي بالأنظمة

لنفترض أنك تريد من وكيلك أن يقرأ مواعيدك، أو ينشئ صفحة في Notion، أو يبحث في قاعدة بيانات شركتك. هذه الأشياء تعيش خارج الطبقة التشغيلية، والوكيل يحتاج إلى وصلة تصل إليها.

يمكنك بناء أداة خاصة لكل خدمة باستخدام الـAPI. لكن إن اضطر كل منتج وكل خدمة إلى اختراع وصلة مختلفة، ستتكرر المشكلة عشرات المرات.

هنا يأتي **MCP، أو Model Context Protocol**: طريقة موحدة تستطيع بها خدمة خارجية أن تعرض للوكيل الأدوات والبيانات المسموح له باستخدامها.[7]

في الصورة الذهنية، تخيّل أن غرفة الوكيل امتد منها ذراع إلى التقويم. هذا الذراع لا يعطيه التقويم كله بلا حدود؛ بل يضع أمامه إجراءات واضحة، مثل: «اقرأ مواعيد اليوم»، و«أنشئ موعداً جديداً»، و«ابحث عن وقت متاح».

حين تقول: «احجز لي اجتماعاً غداً»، يرى الوكيل هذه الأدوات، ويختار أداة إنشاء الموعد، ويرسل لها التفاصيل. الوصلة تنفذ الإجراء في خدمة التقويم، ثم تعيد النتيجة إلى الوكيل.

الـMCP هو الوصلة التي تعرض للوكيل أفعالاً وبيانات واضحة داخل خدمة خارجية. ولا يرى الوكيل أو يفعل إلا ما عرضته الوصلة وسمحت به الصلاحيات.

وقد تستخدم وصلة MCP واجهة API في داخلها. ببساطة: **الـAPI هو الباب البرمجي الذي فتحته الخدمة، وMCP طريقة منظمة تجعل قدرات ذلك الباب ظاهرة ومفهومة للوكيل.**

قاعدة البيانات: معلومات يمكن الرجوع إليها والعمل عليها

إذا كانت معلومات العمل موزعة في صور، ومحادثات، وملفات عشوائية، يصعب على الوكيل الوصول إلى الحقيقة الحالية.

قاعدة البيانات تحفظ المعلومات في بنية يمكن الاستعلام عنها وتحديثها. وهذا يتيح للنظام، ضمن صلاحياته، أن يسأل مثلاً:

- ما الطلبات المفتوحة؟
- من العملاء الذين لم يتلقوا متابعة؟
- ما آخر حالة لهذا المشروع؟
- وما السجلات التي تحتاج إلى تحديث؟

لكن الاتصال بقاعدة البيانات لا يعني منح حق التعديل الكامل. يمكن أن يكون الوصول للقراءة فقط، أو لجداول محددة، أو عبر عمليات مصممة مسبقاً. مرة أخرى: **الوصول والصلاحية طبقتان منفصلتان.**

ثالثاً: تقسيم العمل وتشغيله

Sub-agent: عامل في سياق منفصل

تخيّل أن مهمة البحث أعادت عشرات الصفحات والسجلات. إذا قرأها الوكيل الرئيسي كلها، قد تمتلئ جلسته بالضوضاء، بينما ما يحتاجه فعلاً هو خلاصة صغيرة.

يمكنه تفويض مهمة البحث إلى **وكيل فرعي Sub-agent**.

الوكيل الفرعي هو وكيل منفصل تُسند إليه مهمة محددة. يعمل عادةً في نافذة سياق مستقلة، وقد يحمل تعليمات أو أدوات أو صلاحيات أو حتى نموذجاً مختلفاً، ثم يعيد النتيجة أو الملخص إلى الجهة التي فوضته.[8]

أهم فوائده:

1. عزل السياق

يقرأ المواد الثقيلة في سياقه، ويعيد الخلاصة بدلاً من كل الضوضاء.

2. التخصص

يمكن أن يكون مراجعاً، أو باحثاً، أو مختبِراً، أو مدققاً أمنياً، مع تعليمات وأدوات تناسب مهمته.

3. التوازي

يمكن لعدة وكلاء مستقلين بحث جوانب مختلفة في الوقت نفسه، إن كانت البيئة تدعم ذلك.

لكن الوكيل الفرعي ليس قوة مجانية إضافية. هو يستهلك وقتاً وتوكنز، وقد يعيد نتيجة ناقصة، وقد يحتاج إلى معلومات لم ينقلها إليه الوكيل الرئيسي. استخدمه عندما تحتاج فعلاً إلى عامل متخصص أو إلى تنفيذ أجزاء مستقلة في الوقت نفسه.

Automation: حين لا تعيد تشغيل العملية يدوياً

لنفترض أنك كلما استلمت ملفاً جديداً تفتحه، وتقرأه، وتكتب خلاصة عنه، ثم تحفظ الخلاصة في مكان محدد. بعد عدة مرات ستلاحظ أنك تعيد العملية نفسها.

عندما نربط العملية بمحفّز يبدأها تلقائياً، فتُنفَّذ خطواتها من دون أن نعيد تشغيلها يدوياً كل مرة، نقترب من **الأتمتة Automation**.

قد يبدأ المحفّز بسبب:

- موعد زمني؛
- وصول ملف؛
- تسجيل نموذج؛
- تحديث في قاعدة بيانات؛
- وصول بريد؛
- أو ضغط زر واحد.

الأتمتة لا تتطلب وكيلاً دائماً.

أتمتة ثابتة

الخطوات محددة مسبقاً: إذا حدث كذا، نفذ أ ثم ب ثم ج.

أتمتة وكيلة

يبدأ المحفّز العملية، ثم يقرر النموذج بعض الخطوات وفق الحالة والنتائج.

هنا يظهر فرق مفيد:

> **الأتمتة تجيب: كيف تبدأ العملية وتعمل دون إعادة تنفيذها يدوياً؟**
>
> **الوكيل يجيب: من يقرر الخطوة التالية عندما لا يكون المسار ثابتاً؟**

يمكن أن يكون لديك وكيل تشغله يدوياً، فلا تكون العملية مؤتمتة بالكامل. ويمكن أن يكون لديك Workflow مؤتمت لا يستخدم أي نموذج. ويمكن جمع الاثنين.

الخريطة الحاسمة

العنصر	ما الذي يضيفه؟	السؤال الذي يحله
Tool	إجراء تنفيذي	ما الإجراء الذي لا يستطيع النظام تنفيذه الآن؟
Skill	طريقة ومعيار قابلان لإعادة الاستخدام	ما الذي أكرر شرح طريقته؟
API	باب برمجي لخدمة	كيف تطلب أداة قدرة أو بيانات من خدمة أخرى؟
MCP	وصلة موحدة للأدوات والموارد	كيف نعرض قدرات أنظمة خارجية للوكيل بطريقة مفهومة؟
Database	حالة وبيانات منظمة	أين تعيش المعلومات التي يجب الاستعلام عنها وتحديثها؟
Sub-agent	عامل وسياق منفصل	ما المهمة التي تستفيد من العزلة أو التخصص أو التوازي؟
Automation	تشغيل متكرر بعد محفّز	ما العملية التي لا ينبغي أن أبدأها يدوياً كل مرة؟
Orchestration	تنسيق بين الأجزاء، ويمكن حفظه داخل Skill	كيف تتحرك الخطوات والأدوات والوكلاء والنتائج معاً؟

خلاصة الفصل

الأداة تمنح الوكيل إجراءً، والSkill تمنحه طريقتك، والAPI أو MCP يصلانه بخدمة أو بيانات خارجية، وقاعدة البيانات تحفظ حالة العمل، والوكيل الفرعي يتولى جزءاً مستقلاً، والأتمتة تجعل العملية تبدأ وتتكرر.

9 من الفهم إلى التمكّن

أصبحت الخريطة الآن مكتملة. أنت تعرف ما النموذج، وكيف يصل إليه كلامك، وما نافذة السياق، وكيف تمنحه الأدوات قدرة على التنفيذ، ولماذا تتغير قدرته عندما ينتقل من محادثة في المتصفح إلى بيئة تعمل على مشروعك.

لكن فهم الخريطة ليس هو التمكّن.

التمكّن يبدأ عندما تستخدم هذه المبادئ في عمل حقيقي: تعطي الوكيل طلباً، وترى النتيجة، وتفهم لماذا أخطأ، ثم تحسّن الجزء الذي يحتاج إلى تحسين. هناك فهم لن يتكوّن عندك من القراءة وحدها؛ يتكوّن عندما تجرّب، وتخطئ، وتصحح، وتعيد المحاولة.

تمرين واحد يجمع الدليل كله

اختر مهمة حقيقية من عملك تستطيع تقييم نتيجتها. قد تكون إعداد تقرير، أو تحليل ملفات، أو تنظيم معلومات مشروع، أو بناء أداة صغيرة.

قبل أن تبدأ، أجب عن خمسة أسئلة:

1. ما النتيجة التي أريدها؟

ما الذي يجب أن يوجد في النهاية: ملف، أم تحليل، أم قرار، أم تغيير داخل مشروع؟ وكيف يجب أن يبدو؟

2. ما المعلومات التي يحتاج إليها؟

ما الملفات أو المواقع أو البيانات التي يجب أن يقرأها؟ وهل توجد تعليمات أو أمثلة سابقة توضّح ما تريده؟

3. ما الإجراءات التي يحتاج إليها؟

هل يحتاج إلى البحث، أم قراءة ملفات وكتابتها، أم تشغيل مشروع، أم الاتصال بخدمة خارجية؟

4. ما الحدود؟

ما الذي يستطيع قراءته أو تغييره؟ وما الذي يجب أن يتوقف قبله ويطلب موافقتك؟

5. كيف ستتحقق من النتيجة؟

ما الذي ستراجعه؟ ما الاختبار الذي يجب أن ينجح؟ ومتى تعتبر المهمة مكتملة؟

ثم ابدأ:

> **نفّذ ← راقب النتيجة ← حدّد موضع المشكلة ← حسّن جزءاً واحداً ← أعد المحاولة**

قد تكون المشكلة في طلبك، أو في نقص السياق، أو في أداة غير موجودة، أو في صلاحية لم تمنحها، أو في حاجة الوكيل إلى مهارة تشرح له طريقتك. مع كل محاولة ستفهم الوكلاء أكثر. وهذه الخبرة لا تُبنى بالحفظ، بل بالعمل.

إذا لم تتذكر إلا عشر جمل

1. **الشركة ليست المنتج، والمنتج ليس النموذج.**
2. **النموذج هو العقل الذي يفهم ويولّد، لكنه لا ينفذ في العالم وحده.**
3. **كلامك يدخل إلى النموذج على شكل توكنز.**
4. **نافذة السياق هي الحد الأقصى لما يستطيع النموذج الرجوع إليه الآن.**
5. **المعلومة الموجودة في ملف لا يعرفها النموذج حتى تصل إلى سياقه.**
6. **الأداة تمنح النظام إجراءً يستطيع تنفيذه.**
7. **الطبقة التشغيلية تحدد ما يراه النموذج، وما الأدوات والصلاحيات التي يملكها.**
8. **قد يعمل النموذج نفسه بصورة مختلفة تماماً داخل بيئتين مختلفتين.**
9. **الوكيل نظام يعمل نحو هدف باستخدام الأدوات داخل حلقة من التنفيذ والمراجعة.**
10. **التمكّن لا يأتي من حفظ أسماء الأدوات، بل من استخدامها في أعمال حقيقية.**

ما الذي يأتي بعد هذا الدليل؟

هذا الدليل شرح لك طبقة الذكاء الاصطناعي: النموذج، والسياق، والأدوات، والطبقة التشغيلية، والوكيل.

أما بناء التطبيقات والأتمتة والأنظمة، فيحتاج إلى خريطة ثانية للجانب البرمجي: ما التطبيق؟ أين تعيش البيانات؟ كيف تتصل أجزاؤه؟ وكيف ينتقل من ملفات على جهازك إلى شيء يعمل ويستخدمه الآخرون؟

سنفكك تلك الخريطة في دليل مستقل. لا تحتاج أن تصبح مبرمجاً تقليدياً قبل أن تبدأ، لكنك تحتاج إلى فهم الأرض التي يعمل فيها وكيلك.

خاتمة: لقد اختفى السحر، وبقيت القدرة

قد تبدو ChatGPT وClaude وGemini وClaude Code وCodex أدوات لا يجمع بينها شيء. لكنك تعرف الآن ما يوجد تحت أسمائها.

هناك نموذج يعمل داخل طبقة تشغيلية. يصل إليه سياق محدد. وتُعرض عليه أدوات تسمح له بتنفيذ إجراءات ضمن صلاحيات وحدود. وعندما يقرأ نتائج ما نفّذه ويقرر الخطوة التالية، يصبح النظام وكيلاً قادراً على إنجاز عمل كامل.

من الآن فصاعداً، لن تحتاج إلى تعلّم عالم جديد مع كل أداة تظهر. ستفككها، وتعرف ما الذي تراه، وما الذي تستطيع فعله، وأين تعمل، ثم تقرر إن كانت تناسبك.

وهنا يبدأ التمكّن الحقيقي.

إذا أردت أن تتعلّم من خلال عملك أنت

أعطاك هذا الدليل الخريطة، لكن القدرة تُبنى أثناء التنفيذ.

في **برنامج التحوّل 1:1** نبدأ من عملك أو مشروعك أنت، لا من تمارين عامة. نعمل معاً، وأنت الذي ينفذ ويتعلّم: تختار البيئة المناسبة، وتدير السياق، وتستخدم الأدوات، وتبني ما يحتاج إليه عملك من تطبيقات وعمليات أتمتة ووكلاء.

الهدف ألا تخرج بنتيجة واحدة فقط، بل أن تصبح قادراً على بناء هذه الأشياء وتطويرها بنفسك.

ملحق: قاموس سريع

هذا القاموس للمراجعة، لا للحفظ. ارجع إليه حين تختلط عليك الطبقات.

المصطلح	المعنى المبسط
Provider — المزوّد	الشركة التي تطور النماذج أو تتيح استخدامها، مثل OpenAI وAnthropic وGoogle.
Product — المنتج	التطبيق الذي تستخدمه، مثل ChatGPT أو Claude Code. قد يعمل داخله نموذج واحد أو أكثر.
Model — النموذج	العقل الذي يفهم المدخلات ويولّد المخرجات. هو جزء من الأداة التي تستخدمها، وليس المنتج كله.
Prompt — الطلب	الكلام أو التعليمات التي ترسلها إلى النموذج.
Token — توكن	الوحدة التي تُقاس بها كمية ما يدخل إلى النموذج وما يخرج منه. يُستخدم عدد التوكنز لحساب سعة نافذة السياق والتكلفة.
Context — السياق	كل المعلومات الموجودة أمام النموذج في هذه اللحظة: الطلب، والمحادثة، والتعليمات، والملفات التي قرأها، ونتائج الأدوات.
Context Window — نافذة السياق	الحد الأقصى من التوكنز التي يستطيع النموذج الرجوع إليها في المرة الواحدة.
System Instructions — تعليمات النظام	التعليمات الأساسية التي تضعها الجهة التي بنت المنتج لتحدد دور النموذج وسلوكه وحدوده.
Memory — الذاكرة	معلومات يحفظها النظام خارج الجلسة، ثم يعيد المناسب منها إلى السياق لاحقاً.
State — الحالة	المكان الذي وصل إليه العمل الآن: ما أُنجز، وما الملفات الناتجة، وما الخطوة التالية.
Tool — الأداة	إجراء يستطيع النظام تنفيذه، مثل البحث، أو قراءة ملف، أو كتابة ملف، أو إرسال بريد.
Tool Calling — استدعاء الأداة	أن يطلب النموذج استخدام أداة، فتنفذها الطبقة التشغيلية ثم تعيد النتيجة إليه.
Permission — الصلاحية	ما يُسمح للنظام بقراءته أو تغييره أو تنفيذه، وضمن أي حدود.
Harness — الطبقة التشغيلية	البرنامج المحيط بالنموذج؛ يجمع له السياق، ويعرض عليه الأدوات، وينفذ طلباتها، ويدير الصلاحيات واستمرار العمل.
Environment — بيئة العمل	المكان الذي يعمل فيه النظام والموارد التي يصل إليها، مثل المتصفح، أو جهازك، أو مجلد مشروع، أو خادم.

المصطلح	المعنى المبسط
Agent — الوكيل	نظام يستخدم نموذجاً وأدوات للوصول إلى هدف، ويراجع نتائج ما فعله ويقرر الخطوة التالية.
Agentic Loop — الحلقة الوكيلية	تكرار فهم الهدف، واختيار الخطوة، والتنفيذ، وقراءة النتيجة، ثم تعديل المسار حتى تكتمل المهمة.
Workflow — سير العمل	خطوات مرتبة تنقل العمل من البداية إلى النتيجة.
Skill — المهارة	تعليمات وموارد تحفظ طريقة أداء عمل ما، كي يستطيع الوكيل تكرارها من دون أن تشرحها من جديد كل مرة.
API — واجهة برمجة التطبيقات	باب برمجي يسمح لبرنامج بطلب بيانات أو تنفيذ إجراء داخل خدمة أخرى.
API Key — مفتاح API	مفتاح سري تستخدمه الخدمة للتحقق من أن الاتصال مسموح. لا يُلصق في المحادثات ولا يُنشر مع ملفات المشروع.
`.env` **— ملف المتغيرات السرية**	ملف شائع لحفظ مفاتيح الاتصال خارج الكود، كي تستخدمها الأدوات من الخلفية. يجب ألا يُرفع إلى GitHub أو يُشارك مع الآخرين.
MCP — Model Context Protocol	طريقة موحدة لعرض أدوات وبيانات الخدمات الخارجية لتطبيقات الذكاء الاصطناعي.
Database — قاعدة البيانات	مكان منظم لحفظ معلومات يمكن البحث فيها وقراءتها وتحديثها.
Sub-agent — وكيل فرعي	وكيل منفصل يتولى جزءاً محدداً من العمل، ثم يعيد نتيجته إلى الوكيل الرئيسي.
Automation — الأتمتة	عملية تبدأ وتنفذ تلقائياً عند موعد أو حدث أو محفّز محدد.
Orchestration — التنسيق	تنظيم انتقال العمل والنتائج بين عدة مراحل أو أدوات أو وكلاء.
Hallucination — الهلوسة	جواب يبدو مقنعاً، لكنه غير صحيح أو غير مدعوم بمصدر كافٍ.

ملحق: أفكار شائعة تحتاج إلى تصحيح

الفكرة الشائعة	التصحيح
«اسم التطبيق هو اسم النموذج.»	قد يكون التطبيق منتجاً يشغّل نموذجاً واحداً أو عدة نماذج. افصل الشركة والمنتج والنموذج.
«النموذج يعرف آخر ما حدث على الإنترنت.»	النموذج لا يملك حداثة مضمونة. يحتاج إلى أداة ومصدر حديثين عندما تتطلب المهمة معلومات متغيرة.
«الملف موجود على الجهاز، إذاً النموذج يعرفه.»	لا يعرفه حتى تدخله البيئة في السياق، غالباً بعد أن تطلب أداة قراءته.
«نافذة سياق كبيرة تعني ذاكرة دائمة.»	النافذة مساحة عمل للدورة؛ الدوام يحتاج إلى آلية حفظ واسترجاع خارجية.
«كلما أرسلت معلومات أكثر صارت النتيجة أفضل.»	المعلومات غير المرتبطة قد تدفن المهم وتزيد التشويش. الصلة والتنظيم أهم من الكمية وحدها.
«النموذج شغّل الأداة بنفسه.»	النموذج طلب الاستدعاء؛ التطبيق أو الخادم نفذ الفعل وأعاد النتيجة.
«منح الوصول يعني منح حق التعديل.»	يمكن فصل الاتصال عن القراءة، والقراءة عن الكتابة، والكتابة عن الحذف أو النشر.
«API هي المفتاح السري.»	الـAPI هي الباب البرمجي للخدمة؛ أما مفتاح API فهو السر الذي قد تسمح به الخدمة للاتصال، ويجب حمايته.
«MCP بديل لكل API.»	MCP قد يعرض قدرات مبنية على APIs بطريقة موحدة؛ الاثنان قد يعملان معاً.
«كل وكيل أتمتة، وكل أتمتة وكيل.»	قد تشغل وكيلاً يدوياً، وقد تبني أتمتة ثابتة لا تستخدم نموذجاً أصلاً.
«الوكيل الفرعي ذكاء مجاني إضافي.»	هو عامل منفصل يستهلك وقتاً وتوكنز، وفائدته أن يتولى عملاً متخصصاً أو مستقلاً عن الوكيل الرئيسي.
«استخدام النموذج نفسه يعني الحصول على النتيجة نفسها.»	السياق والأدوات والبيئة والتعليمات والصلاحيات والحلقة تغير السلوك العملي جذرياً.
«إذا أنجز الوكيل المهمة مرة، فهي جاهزة للأتمتة.»	النجاح مرة لا يكفي. جرّب الحالات المختلفة، وحدد كيف تتحقق من النتيجة وماذا يحدث عند الفشل قبل تشغيلها تلقائياً.
«الوكيل القوي لا يحتاج إلى مراجعة.»	كلما ارتفع أثر الفعل، زادت أهمية القيود والسجلات والاختبارات ونقاط موافقة الإنسان.

المصادر والمطالعة الإضافية

هذا الدليل يبسّط المفاهيم لأغراض عملية. للتوسع التقني، ارجع إلى المصادر الأصلية التالية:

2 Anthropic — Context windows: شرح نافذة السياق وما يدخل فيها أثناء الطلب.

3 Anthropic — How tool use works: دورة طلب الأداة وتنفيذها وإعادة نتيجتها إلى النموذج.

4 Anthropic — Trustworthy agents in practice: طبقات النظام الوكيلي وحلقة التخطيط والفعل والملاحظة والتعديل.

5 Anthropic — Agent Skills overview: المهارات كتعليمات وموارد قابلة لإعادة الاستخدام.

6 MDN Web Docs — API: تعريف مبسط لواجهة برمجة التطبيقات.

7 Model Context Protocol — Introduction وArchitecture: تعريف المعيار وبنيته ومكوّناته.

8 Claude Code Docs — Subagents: مثال عملي على الوكلاء الفرعيين، وعزل السياق، وتخصيص الأدوات والنماذج.

9 OpenAI Help — What are tokens and how to count them?: شرح التوكنات، وعلاقتها بالكلمات والتكلفة وحدود النماذج.

10 OpenAI — GPT-3.5 Turbo Instruct: مثال رسمي لنافذة 4,096 توكن في نموذج من الجيل الأقدم، وAnthropic — 100K Context Windows: إعلان نافذة 100 ألف توكن في مايو 2023.

11 Anthropic — Claude 2.1: إعلان نافذة 200 ألف توكن في نوفمبر 2023.

12 Google — Gemini 1.5 and the one million token context window: إعلان نافذة المليون في فبراير 2024، وGoogle AI for Developers — Long context، وAnthropic — Context windows: توثيق النوافذ الطويلة الحالية في أكثر من عائلة من النماذج.

13 أمثلة المنتجات مأخوذة من صفحاتها الرسمية: OpenAI — ChatGPT work and Codex، وAnthropic — Claude Code overview، وGoogle — Gemini، وGoogle Antigravity، وOpenCode.

14 Claude Code Docs — Explore the context window وInteractive mode — built-in commands: توثيق الأمرين `/context` و `/memory` وما يعرضانه.