🚀 دليل توفير 65–80% من Claude Pro

22 نقطة عملية مع أمثلة واقعية — X Mentorship | أحمد عماد

1) رفع PDF خام مقابل تحويله لـ md

المشكلة

لما بترفع PDF خام لـ Claude، كل صفحة بتاخد ~3,000 token بسبب الصور والـ metadata. ملف 20 صفحة = 60,000 token قبل ما تكتب كلمة واحدة.

الحل

حوّل الـ PDF لـ Markdown أو نص عادي قبل الرفع. النص الخالص بياخد ~200 token للصفحة — **توفير فوري 93%.**

الطريقة 1 — Google Docs (مجاني)

1. افتح drive.google.com وارفع الـ PDF
2. كليك يمين ← **Open with** ← **Google Docs**
3. من القائمة: **File** → **Download** → **Plain Text (.txt)** أو **Markdown (.md)**
4. ارفع الـ .txt أو الـ .md لـ Claude بدل الـ PDF

الطريقة 2 — أدوات أونلاين مجانية

- smallpdf.com → PDF to Word → انسخ النص
- ilovepdf.com → PDF to TXT

مثال واقعي

❌ **الغلط:** ترفع "contract_50pages.pdf" مباشرة → *50 × 3,000 = 150,000 token*

✅ **الصح:** تحوّل الـ PDF لـ md وحجمه بقى 10,000 token → *توفير 93%*

> ⚠️ *لو الـ PDF فيه جداول أو مخططات مهمة، اسكان الجزء النصي بس وابعت الصور كـ screenshots منفصلة.*

2) تحديد scope الملفات في Cowork

المشكلة

Cowork بيبني ملفات حسب حاجته هو — بيعمل مجلدات وملفات احتياطية إنت مش طلبتها وده بياكل context.

الحل

حدد الـ scope بوضوح في أول chat. بلّغ Claude بالضبط إيه اللي محتاجه قبل ما يبدأ يشتغل.

Template أول رسالة في Cowork

```
المهمة: [اوصف المهمة]
الملفات المطلوبة فقط:
- [اسم الملف 1]: [وظيفته]
- [اسم الملف 2]: [وظيفته]
لا تعمل أي ملفات إضافية غير اللي ذكرتها.
```

❌ **الغلط:** "ابني لي تطبيق todo list" → Claude هيعمل 7 ملفات معظمها مش محتاجها

✅ **الصح:** حدد index.html + app.js فقط → context نظيف، tokens أقل

3) طول الـ Prompts

المشكلة

ناس بتكتب prompts من 300-500 كلمة وهي مش محتاجة. كل كلمة زيادة في الـ prompt = tokens زيادة في كل رسالة.

الحل

29 كلمة كافية في معظم الأحيان. الـ prompt المثالي: **"عايز [المهمة] عشان [الهدف]."** وخلّي Claude يسألك لو محتاج تفاصيل.

```
عايز [المهمة] عشان [الهدف].
لو عندك أسئلة توضيحية، اسألني قبل ما تبدأ.
```

❌ **الغلط:** 450 كلمة تشرح فيها كل التفاصيل من البداية

✅ **الصح:** "عايز خطة لإدارة مشروع تقني متأخر عشان أوصله في الـ deadline. اسألني الأسئلة اللي محتاجها الأول" → *توفير 94%*

> *متى تكتب prompt طويل؟ فقط لو المهمة تقنية ومحتاجة specs محددة جدًا، أو عندك قيود صارمة من framework أو style guide.*

4) إرسال رسائل منفصلة لكل مهمة

المشكلة

لما بتبعت رسالة فيها 3 مهام منفصلة، Claude بيحتاج يقرا كل الـ context لكل مهمة — ده بيضاعف استهلاك الـ tokens.

الحل

رسالة واحدة = مهمة واحدة. اطلب كل حاجة في رسالة منفصلة جوا نفس الـ chat.

❌ **الغلط:** "1. لخصلي النص ده 2. اقتراحلي عنوان 3. اعمل outline" — كل ده في رسالة واحدة

✅ **الصح:** رسالة 1: "لخصلي النص ده" ← انتظر الرد ← رسالة 2: "اقتراحلي عنوان" ← انتظر الرد ← رسالة 3: "اعمل outline"

في الطريقة الأولى، Claude بيحسب الـ context الكامل 3 مرات. في الطريقة التانية، كل رسالة بتاخد context أقل — **توفير 40-60%.**

5) تعديل الرسالة الأصلية بدل Regenerate

المشكلة

لما بتعمل Regenerate أو بتكتب "مش كده، اعمل تاني"، Claude بيقرا كل الـ history من الأول + رسالتك الجديدة. زيادة tokens بدون فايدة.

الحل

اضغط **Edit** ✏️ على رسالتك الأصلية، صلّح فيها، واضغط **Save & Submit**. الـ history هيتبدل ومش هيتراكم.

❌ **الغلط:** أنت: "اكتبلي email" ← Claude يكتب ← أنت: "مش كده" ← Claude يكتب تاني ← أنت: "لأ، باللغة العربية" — *3 رسائل + 3 ردود = 3 × tokens*

✅ **الصح:** اضغط Edit على رسالتك الأصلية وحددها كاملة من أول ← *رسالة 1 + رد واحد = نص الـ tokens — توفير 50-70%*

6) اختيار الموديل الصح

المشكلة

ناس بيستخدموا Opus لكل حاجة. Opus تقريبًا 5x أغلى من Sonnet. لو 70% من مهامك بسيطة، إنت بتدفع 5x زيادة.

دليل الاختيار

الموديل	استخدمه لـ
Haiku	الأسئلة السريعة، التلخيص البسيط، الترجمة، الردود القصيرة
Sonnet	الكتابة، التحليل، الكود المتوسط، معظم المهام اليومية
Opus	التفكير المعقد، الاستراتيجية، القرارات الصعبة، الكود المتقدم
Opus + Extended Thinking	المسائل الرياضية الصعبة، التحليل العميق

كيف تغير الموديل؟ في أعلى الـ chat، اضغط على اسم الموديل الحالي واختار المناسب للمهمة — ممكن تغير في أي وقت جوا نفس الـ chat.

7) إلغاء الملفات الاحتياطية غير الضرورية في Cowork

المشكلة

Cowork يرمي ملفات "احتياطية" في كل مهمة — logs، cache files، backup versions — وكلها بتاكل tokens وبتشوش الـ context.

الحل

حدد الملفات اللي المهمة محتاجاها فعلاً في بداية الـ session.

```
قواعد للـ session دي:
1. لا تعمل ملفات logs
2. لا تعمل backup files
3. لا تعمل README ما لم أطلبه صراحةً
4. الملفات المطلوبة فقط: [قائمة الملفات]
```

النتيجة: بدل 8 ملفات (~12,000 token)، بتاخد 2 ملفات (~3,000 token) — **توفير 75%**

8) إدارة طول الـ Chat

المشكلة

Context في الـ chat الواحد بيكبر بشكل quadratic مش linear — كل رسالة جديدة بتقرأ كل الرسايل اللي فاتت. chat بـ 40 رسالة بياخد 40 × context مقارنة بـ chat جديد.

الحل

كل 15-20 رسالة: لخص الـ brief وافتح chat جديد.

```
ده في 5-7 نقاط session لخّصلي أهم نقط الـ
جديد وأبدأ من نفس النقطة chat بحيث أقدر أفتح
```

بعد ما تاخد الملخص، افتح chat جديد والصق الملخص كأول رسالة مع المهمة الجاية — **توفير 50-70%.**

9) فصل المواضيع في chats مختلفة

المشكلة

لما بتحط 3 مواضيع مختلفة في chat واحد، Claude بيعيد قراءة كل الـ context الميت وهو بيشتغل على الموضوع الجديد. tokens ضايعة.

الحل

موضوع واحد = chat واحد.

- Chat للكود والتقنية
- Chat للكتابة والمحتوى
- Chat للتحليل والبحث
- Chat لكل project مستقل

قبل ما تبدأ أي موضوع جديد: هل الموضوع مرتبط بالـ chat الحالي؟ نعم → كمّل | لأ → افتح chat جديد — **توفير 50-70%**

10) تقليص ملف الـ about-me

المشكلة

ملف about-me بـ 22,000 كلمة بيتبعت في **كل رسالة** في كل session. لو عندك chat بـ 20 رسالة، الـ 22,000 كلمة بتتكرر 20 مرة.

الحل

قلّص الـ about-me لأقل من 2,000 كلمة. وفي آخر كل session، اكتب ملاحظاتك في session-notes.md منفصل.

Template الـ about-me المثالي

```
الاسم: [اسمك]
الدور: [وظيفتك]
اللغة المفضلة: [عربي/إنجليزي/مختلط]
أسلوب الرد: [مختصر/تفصيلي/نقاط/سرد]
سياق مهم:
- [نقطة 1]
- [نقطة 2]
- [نقطة 3]
```

التوفير: من 22,000 × كل turn لـ 2,000 × كل turn — توفير 90%

11) إيقاف Search والـ Connectors الافتراضية

المشكلة

Search والـ Connectors شغّالين by default. كل رسالة بتكلّف tokens إضافية للـ feature ده حتى لو مش محتاجاه.

الحل

اوقف كل feature افتراضيًا. شغّل كل feature حسب المهمة مش حسب الـ account.

خطوات التنفيذ في claude.ai

1. اضغط على ⚙ Settings
2. روح لـ Features أو Integrations
3. وقّف: Web Search، Google Drive Connector، Code Execution
4. شغّل أي feature بس لما تحتاجه في المهمة المحددة

Feature	شغّله لما
Web Search	تحتاج معلومات بعد 2024 أو أخبار حديثة
Google Drive	تشتغل على ملفات Drive مباشرة
Code Execution	تحتاج تشغّل كود وتشوف النتيجة

توفير: 10-20% من overhead كل رسالة

12) استخدام Projects بدل رفع نفس الملف مرات

المشكلة

لو بترفع نفس الـ PDF أو الـ codebase في 5 chats مختلفة، إنت بتحرق 5 × tokens على نفس الملف.

الحل

Projects في claude.ai بتخليك ترفع الملف مرة واحدة، وكل chat جوا الـ Project بيشوفه تلقائيًا.

خطوات التنفيذ

1. في claude.ai، اضغط **Projects** من القائمة الجانبية
2. اضغط **New Project** وادّي الـ Project اسم واضح
3. ارفع الملفات الثابتة مرة واحدة (documents، codebase، style guides)
4. ابدأ كل chat جديد من جوا الـ Project — الملفات موجودة تلقائيًا

❌ **بدون Project:** Chat 1 + Chat 2 + Chat 3 = 3 × رفع = 15,000 token على نفس الملف

✅ **مع Project:** رفع مرة واحدة فقط — **توفير 80-90%**

13) ضبط Personal Preferences مرة واحدة

المشكلة

ناس بتكتب في كل chat: "ردّ بالعربي"، "استخدم bullet points"، "كن مختصر"... ده tokens ضايعة في كل مرة.

الحل

Settings → Personal Preferences — اكتب كل تفضيلاتك مرة واحدة وهيفضلوا للأبد.

```
اللغة: رد بالعربي دايمًا ما لم أطلب تحديدًا غير كده
الأسلوب: مباشر ومختصر بدون مقدمات
التنسيق: استخدم bullet points للقوائم والنقاط المتعددة
الأمثلة: دايمًا اضرب مثال واقعي على أي نظرية
الطول: الرد يكون بالقدر اللي المهمة تحتاجه
التعامل مع الكود: اشرح الكود بعد ما تكتبه
```

التوفير: 100% على التوضيحات المتكررة في كل chat

14) بناء Prompt Library

المشكلة

بتعيد كتابة نفس الـ prompts من الصفر في كل مرة. مهام زي "حلّل الكود ده"، "اكتبلي email احترافي"، "لخصلي الاجتماع" — كلها بتتكتب من جديد.

الحل

اعمل ملف prompts.md وحفظ فيه الـ prompts الثابتة الـ templates بتاعتها. الجزء المتغير بس هو الـ content اللي بتحطّه.

```
## تحليل كود
"حلّل الكود ده وقولي:
1. المشاكل المحتملة
2. تحسينات الـ performance
3. أي security issues
[الكود هنا]"

## Email احترافي
"اكتبلي email [ودي/رسمي] [الموضوع] عن [الجهة] لـ.
النقاط الرئيسية: [النقاط]"
```

أو افتح Project اسمه "Prompt Library" وارفع فيه ملف الـ prompts كـ reference دائمة — **توفير 70-80% في الوقت**

15) 🆕 استخدام Extended Thinking بحكمة

المشكلة

Extended Thinking بياخد tokens كتير — بيفكر "بصوت عالٍ" قبل ما يجاوب. لو استخدمته على مهام بسيطة، إنت بتدفع 3-5x tokens زيادة بدون فايدة.

الحل

Extended Thinking مفيد جدًا — بس في الأنواع الصح من المشاكل.

✅ استخدمه لـ

- مسائل رياضية أو logic معقدة
- قرارات استراتيجية بين خيارات متعددة ومتضاربة
- debugging كود صعب مش قادر تحله
- تحليل أكاديمي أو بحثي عميق

❌ متستخدمهوش لـ

- كتابة إبداعية أو ترجمة
- تلخيص أو أسئلة معلوماتية بسيطة
- مهام روتينية

توفير: 60-80% على المهام البسيطة بدل ما تستخدم Opus × 3-5x

16) 🆕 تقسيم المهام الكبيرة إلى Subtasks

المشكلة

لما بتعطي Claude مهمة كبيرة جدًا في رسالة واحدة، بيحتاج context ضخم جدًا ويولّد output كبير. ده بيرفع الـ tokens وبتزيد احتمالية الإجابة المتوسطة.

الحل

قسّم المهمة الكبيرة لـ subtasks صغيرة وأعطيها بالتسلسل.

```
"ابني نظام authentication كامل" ← المهمة الكبيرة:

Subtask 1: "اعمل database schema للـ users table"
[انتظر وراجع]
Subtask 2: "اعمل POST /register endpoint"
[انتظر وراجع]
Subtask 3: "اعمل POST /login endpoint مع JWT"
```

توفير: 40-60% — context صغير لكل subtask بدل context ضخم + output كبير

17) 🆕 استخدام XML Tags لتنظيم السياق

المشكلة

لما بتبعت context كبير بدون structure، Claude بيقرأ كل حاجة بنفس الوزن. ده بيشتت انتباهه ويزود الـ tokens المطلوبة للفهم.

الحل

استخدم XML tags لتنظيم الـ context وتحديد أولويات المعلومات.

```
<context>
  الشركة: Beetleware — SaaS solutions
  الموقع: مصر والسعودية
</context>

<task>
اكتب job description لـ Senior Python Engineer
</task>

<requirements>
- خبرة: 5+ سنين
- تقنيات: Python, AWS, PostgreSQL
</requirements>
```

Claude هيركّز على الـ task ويستخدم الـ context كمرجع فقط — **توفير 20-30%**

18) Prompt Caching 🆕 للمستخدمين API

المشكلة

لو بتستخدم Claude API في تطبيق، كل request بيبعت نفس الـ system prompt من جديد. لو الـ system prompt 5,000 token وعندك 1,000 request يوميًا = 5,000,000 token يوميًا على system prompt بس.

الحل

استخدم Prompt Caching في الـ API. الـ cached tokens بتكلف 10% فقط من السعر العادي.

```
import anthropic
client = anthropic.Anthropic()
response = client.messages.create(
    model="claude-opus-4-5",
    max_tokens=1024,
    system=[{
        "type": "text",
        "text": "أنت assistant متخصص في [مجالك]...",
        "cache_control": {"type": "ephemeral"}  # ← هنا السحر
    }],
    messages=[{"role": "user", "content": "سؤال المستخدم هنا"}]
)
```

التوفير: 90% على الـ system prompt في كل request

19) Projects System Prompts

المشكلة

معظم الناس بتستخدم Projects كـ "مجلد ملفات" بس. بس الـ Project System Prompt هو أقوى أداة فيه — وناس كتير مش عارفاها.

الحل

كل Project ممكن يكون عنده **System Prompt ثابت** بيتطبق على كل chat جواه. يعني مش محتاج تكتب السياق والقواعد في كل chat من جديد.

خطوات التنفيذ

1. افتح الـ Project من القائمة الجانبية
2. اضغط على **Project Settings** أو ⚙️
3. في خانة **Instructions** أو **System Prompt** اكتب:

```
السياق:
- المشروع: [اسم المشروع وهدفه]
- التقنيات المستخدمة: [stack]
- القيود: [أي قيود تقنية أو تجارية]

قواعد الرد:
- اللغة: [عربي/إنجليزي]
- الأسلوب: [مختصر/تفصيلي]

افتراضات ثابتة:
- [افتراض 1]
- [افتراض 2]
```

كل chat جوا الـ Project هيبدأ بالسياق ده تلقائيًا — مش محتاج تكتبه من جديد أبدًا.

التوفير: 300-500 token في بداية كل chat × عدد الـ chats في الـ Project

20) إدارة الـ Context Handoff بذكاء

المشكلة

النقطة 8 قالت "افتح chat جديد كل 15-20 رسالة" — بس **إزاي** تعمل الـ handoff ده هو اللي بيفرق. handoff ضعيف = بتبدأ من الصفر. handoff قوي = بتكمل بدون فقدان أي حاجة.

الـ Handoff Template المثالي

في آخر الـ chat القديم، ابعت الرسالة دي:

```
عمل Context Handoff للـ chat الجديد.
اكتبه في الشكل ده بالضبط:

## Context Handoff
### المشروع والهدف
[جملة واحدة]
### القرارات المتخذة (مش قابلة للتغيير)
- [قرار 1]
### الوضع الحالي
[وين وصلنا بالضبط]
### الملفات/الكود الموجود
[قائمة مختصرة]
### الخطوة الجاية
[المهمة اللي هنبدأ بيها في الـ chat الجديد]
```

في أول الـ chat الجديد: الصق الـ Handoff وقول "ابدأ من الخطوة الجاية مباشرة"

التوفير: بدل ما تنقل chat بـ 8,000 token، بتنقل Handoff بـ 400 token — توفير 95%

21) تعديل الـ Artifact بدل إعادة الكتابة

المشكلة

لما Claude بيعمل كود أو document كبير وإنت محتاج تعديل، كتير من الناس بيكتبوا "اعمل التعديل ده" — فـ Claude بيعيد كتابة **كل الـ artifact من الأول**. لو الـ artifact 300 سطر كود وإنت محتاج تغيير 5 سطور، دي token مهدرة.

الـ Pattern الصح

بدل: "عدّل الكود ده يعمل Claude ← "X هيكتب 300 سطر تاني

استخدم:

```
"الحالي، عدّل فقط artifact في الـ:
- في السطر [X] اسمها الـ function ~[N]
- غيّر [السلوك القديم] لـ [السلوك الجديد]
لا تعيد كتابة أي حاجة تانية."
```

لـ Documents والـ Markdown

```
"الحالي، عدّل القسم الثالث فقط document في الـ:
- غيّر الفقرة الأولى لـ [النص الجديد]
- أضف bullet point جديد: [النقطة]
باقي الـ document يفضل زي ما هو."
```

التوفير: من إعادة كتابة 5,000-2,000 token لتعديل 200-500 token — توفير 80-90%

22) الـ Feedback المحدد بدل "اعمل تاني"

المشكلة

"غلط، اعمل تاني" أو "مش كده اللي عايزه" — دي أغلى 4 كلمات ممكن تكتبها. Claude هيقرأ كل الـ context من الأول ويعمل output جديد كامل. ومع ذلك غالبًا النتيجة الجديدة مش أحسن لأن Claude مش عارف بالضبط إيه الغلط.

الـ Feedback Framework

```
❌ "مش كده"
✅ "الجزء [X] المطلوب — غلط [Y] بدل [Z]"

❌ "اعمل تاني"
✅ "احتفظ بـ [الجزء الصح]، عدّل بس [الجزء الغلط]"

❌ "مش عاجبني"
✅ "3 تعديلات محددة:
   1. [تعديل 1]
   2. [تعديل 2]
   3. [تعديل 3]"
```

مثال واقعي — Claude كتب email

✅ **Feedback صح:**

```
"كويس ماعدا الـ email:
1. السطر الأول — طوّله شوية، يبان رسمي أكتر
2. الفقرة التانية — اختصرها في جملتين بس
3. الـ closing — استبدل 'بالتوفيق' بـ 'مع خالص التحية'
احتفظ بيه زي ما هو باقي الـ email."
```

Claude هيعمل 3 تعديلات صغيرة بدل ما يكتب email جديد كامل

التوفير: من output كامل جديد (~1,500 token) لتعديل محدد (~300 token) — توفير 80%

📊 جدول مقارنة التوفير في Tokens

#	النقطة	Tokens قبل	Tokens بعد	نسبة التوفير
1	**رفع PDF خام مقابل تحويله لـ md**	3,000 / صفحة	200 / صفحة	**93%**
2	**تحديد scope الملفات في Cowork**	~8 ملفات (~12,000 token)	~2 ملفات (~3,000 token)	**75%**
3	**طول الـ Prompts**	~700 token	~40 token	**94%**
4	**إرسال رسائل منفصلة لكل مهمة**	3 مهام × context كامل	context أقل لكل مهمة	**40-60%**
5	**تعديل الرسالة الأصلية بدل Regenerate**	context × عدد المحاولات	context مرة واحدة	**50-70%**
6	**اختيار الموديل الصح**	تكلفة Opus	تكلفة Sonnet	**80% في التكلفة**
7	**إلغاء الملفات الاحتياطية غير الضرورية في Cowork**	+8,000 token	2,000 token	**75%**
8	**إدارة طول الـ Chat**	context quadratic	context linear	**50-70%**
9	**فصل المواضيع في chats مختلفة**	context كل المواضيع	context موضوع واحد	**50-70%**
10	**تقليص ملف الـ about-me**	22,000 × كل turn	2,000 × كل turn	**90%**
11	**إيقاف Search والـ Connectors الافتراضية**	overhead إضافي لكل رسالة	0 overhead	**10-20%**
12	**استخدام Projects بدل رفع نفس الملف مرات**	رفع × عدد الـ chats	رفع مرة واحدة	**80-90%**
13	**ضبط Personal Preferences مرة واحدة**	~50 token / chat للتوضيح	0	**100% على التوضيحات**
14	**بناء Prompt Library**	كتابة من الصفر كل مرة	نسخ template جاهز	**70-80% في الوقت**
15	🆕 **استخدام Extended Thinking بحكمة**	3-5x × Opus على مهام بسيطة	Haiku/Sonnet	**60-80%**
16	🆕 **تقسيم المهام الكبيرة إلى Subtasks**	context ضخم + output كبير	context صغير لكل subtask	**40-60%**
17	🆕 **استخدام XML Tags لتنظيم السياق**	Claude يقرأ كل حاجة بنفس الوزن	ركّز على الـ task	**20-30%**
18	**Prompt Caching** 🆕 **للمستخدمين API**	system prompt × كل request	10% تكلفة الـ cached tokens	**90% على الـ system prompt**

📊 النقاط المتقدمة (19–22)

#	النقطة	Tokens قبل	Tokens بعد	نسبة التوفير
19	**Projects System Prompts**	400 token context / chat	0 (محفوظ في الـ Project)	**100% على الـ context setup**
20	**إدارة الـ Context Handoff بذكاء**	8,000 token handoff	400 token handoff	**95%**
21	**تعديل الـ Artifact بدل إعادة الكتابة**	2,000-5,000 token	200-500 token	**80-90%**
22	**الـ Feedback المحدد بدل "اعمل تاني"**	output كامل جديد	تعديلات صغيرة	**80%**