La facturation à l'usage en toute confiance

HAMMAD ASLAM

Architecte de solutions cloud senior — Applications IA et conseil au développement

Microsoft

CE QUE NOUS VERRONS AUJOURD'HUI

Au programme d'aujourd'hui

À la fin de cette session, vous comprendrez…

- Le passage à la facturation à l'usage
- Les considérations et contrôles de budget
- Les connaissances fondamentales sur les LLM, les agents et les fenêtres de contexte
- Pourquoi la qualité des agents est le meilleur axe pour optimiser les jetons
- Le contrôle de la qualité et des jetons, et des conseils pratiques d'optimisation

PARTIE 1

Le passage à la facturation à l'usage

TARIFICATION DU FORFAIT DE BASE

Tarif du forfait de base

Copilot Business : 19 $/utilisateur/mois

- 19 $ de crédits IA mensuels

Ce qui ne change pas

- L'expérience de base du développeur reste incluse.
- Les complétions de code et les suggestions de modification suivante restent incluses dans tous les forfaits et ne consomment pas de crédits IA.

LA CHRONOLOGIE

Ce qui change

- **Mai** — Aperçu de la facturation disponible
- **1er juin** — Facturation à l'usage

1. Les unités de requête premium sont remplacées par les crédits IA GitHub.

- 1 crédit IA = 0,01 $ (USD)
- Les crédits IA reposent sur l'usage de jetons selon les tarifs d'API publiés pour chaque modèle.

2. L'usage de Copilot est régi par les crédits disponibles et les contrôles de budget de l'administrateur.

- Aucune expérience de repli, car le modèle 0x n'est plus disponible.

3. Copilot Code Review consommera des minutes GitHub Actions.

- Utilise l'appel d'outils agentique en arrière-plan pour un contexte de dépôt plus large.

FACTURATION À L'USAGE

L'usage repose sur la consommation de jetons

L'usage des crédits IA GitHub repose sur la consommation de jetons

Jetons d'entrée — ce que vous envoyez : invites et nouveau contexte ; peut augmenter avec de gros fichiers ou bases de code.

Jetons de sortie — ce que vous recevez : réponses générées par l'IA et utilisation d'outils ; coût par jeton le plus élevé.

Jetons en cache — ce qui est réutilisé : contexte d'interactions antérieures dans une session ; améliore la vitesse et l'efficacité ; coût par jeton le plus faible.

CE QUE CE CHANGEMENT VOUS APPORTE

Ce que ce changement débloque pour vous

Accès multi-modèles, un seul plan de contrôle

- Continuez d'accéder à plusieurs fournisseurs de modèles via GitHub
- Évitez les contrats et contrôles de facturation distincts
- Gérez l'accès, l'usage et les budgets de façon cohérente entre fournisseurs avec les crédits IA

Usage inclus mutualisé entre utilisateurs — partagez l'usage au sein de votre équipe.

Un ensemble de fonctions plus large

- Des fenêtres de contexte plus grandes
- Amélioration des performances et de la qualité des résultats de Copilot
- Accès aux modèles plus récents et plus capables dès leur sortie
- Futures fonctions agentiques

Pourquoi nous faisons évoluer la tarification

- La facturation à l'usage permet d'offrir des fonctions que l'ancien modèle de tarification ne pouvait pas prendre en charge (par exemple, des contextes plus grands).
- La tarification du calcul IA est basée sur les jetons. La tarification à la requête ne correspond pas aux coûts de calcul IA.
- La facturation à l'usage est conçue pour s'adapter à mesure que les modèles et les tarifs évoluent.

PARTIE 2

Considérations et contrôles de budget

FEUILLE DE ROUTE DES ACTIONS

L'usage varie selon la tâche et le flux de travail

Modèle — les modèles ont des grilles tarifaires et des profils d'usage uniques.

Taille de l'invite — un contexte de mode plus large peut augmenter l'usage de jetons.

Taille de la réponse — des sorties ou un raisonnement plus longs peuvent augmenter l'usage de jetons.

Contexte et cache — un contexte réutilisé peut améliorer l'efficacité.

Flux de travail pilotés par agent — un travail en plusieurs étapes peut traiter plus de jetons.

MCP / Compétences — les outils attachés via les compétences et les serveurs MCP.

Mutualisation des crédits IA

- Chaque licence Copilot apporte sa valeur en crédits IA à un pool d'entreprise unique et partagé.
- Tous les utilisateurs sous licence puisent d'abord dans ce pool.
- Tout usage de crédits IA survenant après la consommation complète du pool de droits constitue une dépense supplémentaire.

Total des crédits IA = Crédits IA des licences + Dépense supplémentaire

Budget au niveau de l'utilisateur (nouveau)

Budgets utilisateur par défaut universels

- Un budget de crédits IA par défaut, par utilisateur, appliqué à tous les utilisateurs
- Garantit que les nouveaux utilisateurs ne dépensent pas trop par accident en fournissant une autorité de dépense de base

Budgets utilisateur individuels — la dérogation pour utilisateurs avancés

- Un budget IA individuel spécifique par utilisateur qui prime sur le budget par défaut universel.
- Contrôle le nombre total de crédits IA qu'un utilisateur peut consommer dans une période de facturation — à la fois l'usage du pool inclus et la dépense supplémentaire.

PARTIE 3

Qualité des agents et optimisation des jetons

Le pari sur les agents n'est plus viable

Quand les jetons sont bon marché, la précision des agents importe peu. Dès qu'ils ne le sont plus, elle exige un travail d'ingénierie.

COMMENT Y PENSER

La qualité influe sur la valeur, qui influe sur le ROI

ROI de l'agent = (Valeur de la sortie de l'agent – Coût en jetons) / Coût en jetons × 100 %

Augmenter la valeur signifie souvent diminuer le coût en jetons relatif.

PRENONS DU RECUL

Le problème de l'erreur cumulée

Les risques qu'un agent se trompe se cumulent rapidement — vrai à la fois pour les boucles internes d'un agent et pour des flux de travail d'agents orchestrés entiers.

- À 95 % de réussite par étape : 95 % → 36 % → 8 % à mesure que les étapes augmentent.
- À 99 % de réussite par étape : 99 % → 82 % → 61 %.

Même à 99 % par étape, un flux de travail de 50 étapes n'atteint qu'environ 60 %.

Plutôt que de compter les jetons, faites en sorte que chaque jeton compte !

PARTIE 4

LLM, agents et fenêtres de contexte

COMMENT Y PENSER

Un LLM est une pure machine à probabilités de mots

À partir de l'invite et du contexte, le modèle prédit le mot suivant le plus probable.

Exemple : « GitHub Copilot est l'outil de développement IA le plus largement… » → le modèle pondère des candidats comme *utilisé* (0,7), *adopté* (0,6), *dépôt* (0,3)… et choisit le plus probable.

Fournissez aussi peu de contexte que possible, mais autant que nécessaire.

BOUCLES À PLUSIEURS ÉTAPES

Travailler avec un agent

Vous et votre projet : invites, fichiers, instructions, compétences, MCP…

L'agent (le harnais) : VS Code Chat, Copilot CLI, agent cloud Copilot, Claude Code, OpenAI Codex…

Le LLM : GPT-5.5, Claude Opus 4.7, Gemini Pro…

Pas de magie — juste du texte ! Le LLM est sans état (stateless).

Fenêtre de contexte et jetons

À chaque boucle, l'entrée comprend l'invite système et les outils, les fichiers, et l'invite ; la sortie est la réponse.

- Jetons d'entrée, jetons de sortie et jetons d'entrée en cache (non garantis).
- Chaque LLM a une limite spécifique de fenêtre de contexte.
- Du 1er au 2e tour, le contexte s'accumule (invite, fichier, réponse, etc.).

PERDU AU MILIEU (LOST IN THE MIDDLE)

La dégradation du contexte (context rot)

Les modèles biaisent les jetons au début et à la fin du contexte.

- **Quand < 50 % des jetons** : ce n'est pas parce que vous pouvez remplir la fenêtre de contexte que vous devriez le faire.
- **Biais de récence** : quand > 50 % des jetons, les modèles biaisent les jetons depuis la fin du contexte.

Référence : https://www.producttalk.org/context-rot/

PARTIE 5

Contrôles de qualité et de jetons

EFFET DES OPTIMISATIONS

De l'ingénieur assisté par l'IA à l'ingénieur IA

Ingénieur assisté par l'IA — travaille avec un seul agent, principalement en synchrone.

Ingénieur IA — orchestrateur de plusieurs agents asynchrones.

LES DEUX PLUS GRANDS LEVIERS

Choix du modèle et mode auto

- Modèles de raisonnement (Opus, GPT-5.5) pour les tâches synchrones comme la planification, l'architecture et le débogage.
- Modèles de milieu de gamme (Sonnet, GPT-5.4) pour l'implémentation asynchrone.
- Bas de gamme (Haiku, GPT-mini) pour les petits refactorings, les tâches répétitives et les mises à jour de documentation.
- Mode Auto comme choix par défaut paresseux (détection de l'intention de la tâche).

Ingénierie du contexte

- Autant que nécessaire, aussi peu que possible. Ne fournissez que le contexte pertinent.
- Le compactage des sessions peut aider, mais peut aussi entraîner la perte d'informations précieuses. À utiliser avec prudence.
- Utilisez /clear souvent, pour chaque nouvelle tâche.

VOTRE INVITE

Votre invite

L'invite système et les outils sont **toujours actifs**.

- Soyez précis.
- Ajoutez une description.
- Ajoutez des signaux d'arrêt. « Arrête-toi si X. »
- Ajoutez le contexte connu à l'avance : fichiers, dossiers, sites web, etc.

RECHERCHER → PLANIFIER → IMPLÉMENTER

Rechercher → Planifier → Implémenter

/research (ex. Gemini 2.5 Pro) — « Je veux changer X. Quels fichiers sont pertinents ? » → produit un raisonnement et des appels de modification.

/plan (ex. Opus 4.7) — transforme l'entrée du plan en une spécification précise.

/fleet (ex. GPT-5.4) — exécute la spécification précise à travers les fichiers.

Contrôles déterministes

Avec tests unitaires : un changement bogué → des tests en échec → un changement de correction → des tests qui passent.

Sans tests unitaires : un changement bogué en entraîne d'autres (session de débogage, incident) — changement bogué 2, 3, 4…

Sans garde-fous : minutes de CI/CD gaspillées, cycles de revue Copilot, temps humain, etc.

CONFIGURATIONS D'AGENT

Configurations d'agent

- **Instructions persistantes** — COPILOT-INSTRUCTIONS.MD
- **Instructions ciblées** — ./.GITHUB/INSTRUCTIONS/*.INSTRUCTIONS.MD
- **Agents personnalisés** — ./.GITHUB/AGENTS/*.AGENT.MD
- **Fichiers d'invite** — ./.GITHUB/PROMPTS/*.PROMPT.MD
- **Compétences** — ./.GITHUB/SKILLS/*/SKILL.MD
- **MCP**, **Sous-agents**, **Mémoire Copilot**

Instructions persistantes

Toujours actives. ./.GITHUB/COPILOT-INSTRUCTIONS.MD ou ./AGENT.MD

Mettez-y :

- les non-négociables de vos projets
- un journal des erreurs récurrentes des agents
- des consignes pour réduire la sortie (« sois concis »)
- Gardez-les très courtes.
- Ne les générez pas avec l'IA.
- Itérez, maintenez et même recréez-les souvent.

Agents personnalisés

Utilisateur : « /tdd-red ajoute un point de terminaison d'API »

- Le harnais récupère le fichier de l'agent.
- Le harnais insère la définition de l'agent personnalisé.
- Le harnais ajuste les outils disponibles.
- Le harnais ajoute l'invite de l'agent personnalisé.

COMPÉTENCES (SKILLS)

Compétences

Utilisateur : « Travaille sur l'API »

- Le harnais met toutes les descriptions de compétences dans le contexte.
- Le LLM au harnais : « J'ai besoin de la compétence API. »
- Le harnais charge la compétence complète dans le contexte.

MCP

Utilisateur : « Lis le ticket #45 »

- Le harnais : « Tu disposes de l'outil MCP get_issue pour lire un ticket. »
- Le LLM : utilise l'outil → lire le ticket #45.
- Le harnais appelle l'API MCP et fournit le ticket #45.

SOUS-AGENTS

Sous-agents

Session principale : « Trouve ma fonctionnalité » → le LLM demande au harnais de « créer un sous-agent ».

- Le sous-agent traite plusieurs documents avec ses propres instructions et invite.
- Le harnais réinsère le résumé dans le contexte principal.

QUOI D'AUTRE

Autres configurations d'agent

Instructions ciblées — instructions conditionnelles basées sur des motifs de chemin de fichier ; non déterministes, offertes à l'agent comme les compétences.

Fichiers d'invite — invites invoquées manuellement ; peuvent réduire l'ensemble d'outils et invoquer des agents personnalisés ; à utiliser comme point de départ manuel.

Mémoire Copilot — instructions automatisées, petites, « toujours actives » ; utilisées uniformément sur les surfaces Copilot ; à vérifier régulièrement.

Conseils pour utilisateurs avancés

Ces conseils nécessitent de bonnes connaissances et du temps, car ils sont conditionnels ou comportent des compromis.

- **Penser en code** — préférez créer des scripts pour analyser des fichiers plutôt que de les donner à l'IA.
- **Comparer CLI et MCP** — les outils CLI peuvent être plus optimaux dans certains cas grâce à moins de jetons statiques.
- **Améliorer les sorties du shell** — les sorties peuvent être très longues ; utilisez un outil comme rtk pour les réduire.
- **Exécuter /chronicle régulièrement** — analysez votre usage dans Copilot CLI pour trouver des axes d'amélioration.
- **Regrouper les appels d'outils** — un plugin peut aider à réunir plusieurs appels d'outils en un seul.
- **Optimisation du contexte spécifique au modèle** — les modèles se comportent différemment et peuvent être ajustés.

QUOI D'AUTRE

Développez vos compétences analytiques

Le codage n'a jamais été la vraie valeur des développeurs — c'étaient leurs compétences analytiques et leur capacité à devenir rapidement compétents dans n'importe quel domaine.

- **Appliquez une bonne architecture** — DDD, architecture hexagonale, CQRS, conception événementielle… ; facilite la découverte par l'agent et donne des garde-fous plus solides.
- **Itérez sur les invites et les configurations d'agent** — abordez les agents IA avec un esprit d'ingénieur ; gardez les configurations à jour et traitez les erreurs d'agent comme des incidents.
- Utilisez `/chronicle` dans la CLI régulièrement pour comprendre les axes d'amélioration.

RÉSUMÉ

5 choses à commencer dès aujourd'hui

- Choisissez le bon modèle pour la bonne tâche. Appuyez-vous sur le mode Auto.
- Fournissez des consignes claires dans vos invites.
- Rechercher – Planifier – Implémenter.
- Fournissez des garde-fous déterministes (tests, linters, analyses de sécurité, etc.).
- Maintenez un copilot-instructions.md concis et rédigé à la main. Utilisez-le comme journal des erreurs d'agent et pour réduire les sorties.

BONUS

Petite revue de l'interface

(Revue de l'interface)

Merci

Diapositive de remerciement

haslam93/GitHub-Copilot---Usage-Based-Billing-Info-Tips