Claude Code sans facture d’API : la stack locale

Claude Code, zéro facture d’API, durablement

Voici le setup exact pour faire tourner Claude Code sur un modèle local, sans facture d’API. Zéro coût. Même flux de travail. Mêmes résultats. Aucun piège caché.

À lire avant de vous lancer

Ce guide n’est pas fait pour tout le monde.

Si votre usage est léger, quelques prompts par jour, l’API Anthropic reste probablement le bon choix. Restez dessus.

Ce blueprint s'adresse à l'opérateur qui :

- dépense entre 60 et 200 dollars par mois en facture d'API Claude ;
- lance de longues sessions agentiques qui consomment des tokens rapidement ;
- travaille sur du code sensible qui ne doit pas quitter sa machine ;
- se déplace et a besoin d'une capacité hors ligne ;
- refuse simplement de payer pour quelque chose qu'il peut faire tourner lui-même.

Si vous vous reconnaissez, continuez la lecture.

Sinon, gardez ce document sous le coude : il vous servira le jour où la facture deviendra assez irritante pour valoir le détour.

Ce qui a réellement changé en janvier 2026

Ollama a publié sa version v0.14 le 16 janvier 2026. Une seule mise à jour, qui change tout.

Avant la v0.14 : faire dialoguer Claude Code avec un modèle local imposait de passer par un intermédiaire (litellm, mitmproxy, claude-code-router, ou un autre middleware) qui cassait toutes les trois semaines.

Après la v0.14 : Ollama expose un endpoint `/v1/messages` qui parle nativement le format de l'API Anthropic.

Claude Code envoie une requête en attendant une réponse au format Anthropic. Ollama l'intercepte, fait tourner votre modèle local, et renvoie la réponse dans le format exact que Claude Code attend.

Pas de middleware. Pas de proxy. Rien à bricoler entre les deux.

Deux variables d'environnement, et vous êtes opérationnel.

Comment ça fonctionne en coulisses

(Vous pouvez sauter cette section si vous voulez juste le setup. Revenez-y si quelque chose casse.)

```
Claude Code envoie une requête au format Anthropic
        ↓
Ollama la reçoit sur le port 11434
        ↓
Ollama fait tourner votre modèle local (glm-4.7, qwen3-coder,
deepseek, etc.)
        ↓
Ollama renvoie une réponse au format Anthropic
        ↓
Claude Code l'affiche comme si rien n'avait changé
```

Claude Code croit dialoguer avec Anthropic. Ollama fait le vrai travail. Votre machine fait l'inférence. Votre carte bancaire ne fait rien.

Prérequis matériels

Réponse honnête : ce point n'est pas négociable.

setup	minimum	idéal	large
ram	16 Go	32 Go	64 Go et plus
vram (gpu)	8 Go	16-24 Go	80 Go
stockage	20 Go libres	50 Go libres	100 Go et plus
os	macos / linux / windows (wsl)	macos apple silicon	linux + nvidia

Concrètement :

- 16 Go de ram + 8 Go de vram : ça marche. glm-4.7-flash tourne correctement, à 15-20 tokens par seconde.
- 32 Go de ram + 16 Go de vram : fluide. qwen3-coder:32b est exploitable, à 20-25 tokens par seconde.
- 64 Go de ram + 24 Go de vram : rapide. deepseek v4-pro file, à 25-35 tokens par seconde.

Note pour Apple Silicon :

Un M2 Pro, M3 ou M4 avec 16 Go de mémoire unifiée fait mieux que la plupart des configurations Nvidia pour cet usage. L'architecture à mémoire unifiée signifie que le GPU et le CPU partagent la ram, donc aucun goulot d'étranglement de vram. Si vous êtes sur Apple Silicon, vous êtes en meilleure posture que vous ne le pensez.

Si votre ordinateur portable devient bruyant pendant l'inférence, c'est normal : ce sont les ventilateurs qui gardent votre machine locale au frais.

Étape 1 : installer Ollama

macOS :

```
# option 1 : télécharger l'application
# rendez-vous sur ollama.com, téléchargez, glissez dans
Applications

# option 2 : homebrew
brew install ollama
```

Linux :

```
curl -fsSL https://ollama.com/install.sh | sh
```

Windows :

Téléchargez l'installateur depuis ollama.com, lancez-le. Ollama démarre automatiquement en tant que service en arrière-plan.

Vérifiez qu'il tourne :

```
ollama --version
# doit afficher 0.14.0 ou une version supérieure
# sinon, mettez à jour avant de continuer
```

Mettez à jour si nécessaire :

```
# macos
brew upgrade ollama

# linux
curl -fsSL https://ollama.com/install.sh | sh
```

Attention : Ollama v0.14.0 est le minimum. Les versions plus anciennes n'exposent pas correctement l'API messages Anthropic. Claude Code va se bloquer ou renvoyer des erreurs 404 si vous êtes sur une version antérieure.

Étape 2 : choisir votre modèle

C'est la décision qui détermine tout. Choisissez mal et l'expérience est mauvaise. Choisissez bien et vous oublierez que vous n'êtes pas sur Claude.

Voici le détail réel.

glm-4.7-flash

Le meilleur compromis. Commencez ici.

```
ollama pull glm-4.7-flash
```

spec	valeur
fenêtre de contexte	128k tokens
vitesse	25 tokens par seconde sur 8 Go de vram
ram requise	~9 Go
appel d'outils	natif
idéal pour	code au quotidien, refactorisations, débogage

Pourquoi c'est le choix par défaut :

- 128k de contexte : il peut tenir l'intégralité de votre base de code dans une seule session.
- Appel d'outils natif : les fonctions agentiques de Claude Code marchent correctement.
- Architecture mixture-of-experts : il tourne sur 16 Go de ram sans forcer.
- Le temps de réponse le plus rapide de la sélection.

La faiblesse honnête : le raisonnement complexe en plusieurs étapes. Si vous lui demandez d'architecturer un système entier de zéro, qwen3-coder s'en sort mieux.

qwen3-coder 32b

Le modèle de raisonnement. À utiliser pour les problèmes difficiles.

```
ollama pull qwen3-coder:32b
```

spec	valeur
fenêtre de contexte	32k tokens
vitesse	12-15 tokens par seconde sur 16 Go de vram
ram requise	~20 Go
appel d'outils	oui
idéal pour	architecture, logique complexe, débogage difficile

Pourquoi la ram supplémentaire en vaut la peine :

- Référence en matière de raisonnement sur code open source en 2026.
- Gère les refactorisations multi-fichiers complexes mieux que glm-4.7.
- Mode de réflexion étendue pour les problèmes réellement difficiles.

La faiblesse honnête : plus lent, plus gourmand en ram, fenêtre de contexte plus petite. Pas idéal pour les allers-retours rapides.

Répartition des cas d'usage :

- code au quotidien → glm-4.7-flash ;
- problème difficile → qwen3-coder 32b.

deepseek v4-pro

Le modèle de pointe, si votre matériel peut le supporter.

```
ollama pull deepseek-coder-v4:pro
```

spec	valeur
fenêtre de contexte	128k tokens
vitesse	20-25 tokens par seconde sur 24 Go de vram
ram requise	~40 Go
livecodebench	93,5 %
idéal pour	code de production, travail sur grande base de code

La réalité honnête :

Ce modèle est réellement de niveau frontière. Le score de 93,5 % sur livecode-bench est réel. Mais il demande du matériel sérieux : 24 Go de vram au minimum pour le faire tourner correctement. Si vous avez un M3 Max ou un M4 Ultra, c'est votre modèle. Sinon, restez sur glm-4.7 ou qwen3.

kimi k2.6

Le modèle multimodal. Pour le travail combiné visuel et code.

```
ollama pull kimi-k2.6
```

spec	valeur
fenêtre de contexte	128k tokens
vitesse	18 tokens par seconde
ram requise	~16 Go
multimodal	oui
idéal pour	design-to-code, analyse d’image, longues tâches agentiques

À utiliser quand vous devez passer des captures d’écran, des maquettes ou des fichiers de design à Claude Code et lui faire générer le code à partir des visuels.

Le guide de décision rapide

```
Vérification matériel :
  16 Go ram + 8 Go vram   → glm-4.7-flash
  32 Go ram + 16 Go vram  → qwen3-coder:32b pour les tâches
difficiles, glm-4.7 au quotidien
  64 Go ram et plus + 24 Go vram et plus → deepseek v4-pro ou
rotation des trois

Vérification tâche :
  refactorisation rapide / correction de bug → glm-4.7-flash
  architecture / logique complexe           → qwen3-coder:32b
  travail visuel / design                    → kimi k2.6
  code de qualité production                  → deepseek v4-pro
```

Étape 3 : les deux variables d’environnement

C’est tout le setup. Deux lignes.

```
export ANTHROPIC_BASE_URL=http://localhost:11434
export ANTHROPIC_AUTH_TOKEN=ollama
export ANTHROPIC_API_KEY=""
```

Techniquement, trois lignes. La troisième (`ANTHROPIC_API_KEY=""`) est indispensable pour empêcher Claude Code de retomber sur une vraie clé d’API si vous en avez une définie ailleurs. Ne la sautez pas.

Rendre cela permanent (pour ne pas avoir à le refaire à chaque session) :

macOS / Linux, à ajouter à la configuration de votre shell :

```
# si vous utilisez zsh (par défaut sur macos)
echo 'export ANTHROPIC_BASE_URL=http://localhost:11434' >>
~/.zshrc
echo 'export ANTHROPIC_AUTH_TOKEN=ollama' >> ~/.zshrc
echo 'export ANTHROPIC_API_KEY=""' >> ~/.zshrc
source ~/.zshrc

# si vous utilisez bash
echo 'export ANTHROPIC_BASE_URL=http://localhost:11434' >>
~/.bashrc
echo 'export ANTHROPIC_AUTH_TOKEN=ollama' >> ~/.bashrc
echo 'export ANTHROPIC_API_KEY=""' >> ~/.bashrc
source ~/.bashrc
```

Ou passez par la configuration `claude.json` (plus propre pour un réglage spécifique à un projet) :

Créez `~/.claude.json` ou `.claude.json` à la racine de votre projet :

```
{
  "env": {
    "ANTHROPIC_BASE_URL": "http://localhost:11434",
    "ANTHROPIC_AUTH_TOKEN": "ollama",
    "ANTHROPIC_API_KEY": ""
  },
  "model": "glm-4.7-flash"
}
```

Cela verrouille le modèle projet par projet. Pratique si vous voulez des modèles différents selon les bases de code.

Étape 4 : lancer Claude Code

Le raccourci en une commande (Ollama v0.14.5 et plus) :

```
ollama launch claude
```

Cela définit automatiquement toutes les variables d'environnement et ouvre Claude Code dans votre répertoire courant. Une commande. C'est fait.

Lancement manuel :

```
# assurez-vous qu'Ollama tourne
ollama serve

# dans un nouveau terminal, démarrez Claude Code
claude --model glm-4.7-flash
```

Vérifiez que ça marche :

Tapez ceci dans le terminal de Claude Code :

```
sur quel modèle tournes-tu ?
```

S'il répond glm-4.7, qwen3-coder, ou n'importe quel modèle que vous avez récupéré, vous y êtes. La facture est désormais à zéro.

Étape 5 : passer sous les 3 secondes de temps de réponse

C'est la partie que la plupart des guides sautent. L'installation brute vous rend opérationnel. Cette section vous rend rapide.

Le goulot d'étranglement est presque toujours l'une de ces trois causes.

Problème 1 : le modèle n'est pas chargé en mémoire

Chaque première requête après un démarrage à froid charge le modèle du disque vers la ram. Cela prend 10 à 30 secondes selon la taille du modèle.

Correction : garder le modèle au chaud

```
# lancez ceci en arrière-plan pour garder le modèle chargé
ollama run glm-4.7-flash ""
```

Ou ajoutez un keep-alive à votre configuration Ollama :

```
# maintient le modèle chargé pendant 1 heure
OLLAMA_KEEP_ALIVE=1h ollama serve
```

Ajoutez-le à la configuration de votre shell pour le rendre permanent :

```
echo 'export OLLAMA_KEEP_ALIVE=1h' >> ~/.zshrc
source ~/.zshrc
```

Problème 2 : ça tourne sur le CPU au lieu du GPU

Si les réponses semblent poussives (sous les 8 tokens par seconde), le modèle tourne sur le CPU, pas sur le GPU.

Vérifiez si le GPU est bien utilisé :

```
# pendant qu'une requête tourne, ouvrez un autre terminal
ollama ps
```

Regardez la colonne `processor`. Elle doit indiquer `gpu` ou `gpu+cpu`. Si elle indique `cpu` uniquement :

```
# vérifiez si Ollama voit votre GPU
ollama info
```

Si le GPU n'apparaît pas :

- nvidia : assurez-vous que les pilotes cuda sont installés et à jour ;
- apple silicon : assurez-vous d'être sur Ollama v0.14 et plus (l'accélération Metal est automatique) ;
- amd : installez les pilotes rocm.

Problème 3 : le modèle est trop grand pour votre vram

Si le modèle ne tient pas dans la vram, il déborde sur la ram et le CPU. Cela tue la vitesse.

Référence rapide des tailles :

modèle	taille sur disque	vram pour un usage full GPU
glm-4.7-flash	~9 Go	8 Go de vram
qwen3-coder:14b	~9 Go	10 Go de vram
qwen3-coder:32b	~20 Go	20 Go de vram
deepseek v4-pro	~40 Go	24 Go et plus de vram

Si votre modèle est trop gros, utilisez la version quantifiée :

```
# au lieu de la pleine précision
ollama pull qwen3-coder:32b

# utilisez la version quantifiée (moitié de la taille, ~90 % de la performance)
ollama pull qwen3-coder:32b-q4_K_M
```

La configuration de réglage des performances

Ajoutez ceci à votre environnement Ollama pour une vitesse maximale :

```
# nombre de requêtes parallèles (augmentez si vous lancez
plusieurs sessions)
export OLLAMA_NUM_PARALLEL=2

# couches GPU (plus élevé = plus d'usage GPU = plus rapide)
# 99 = utiliser le GPU pour tout ce qui est possible
export OLLAMA_GPU_LAYERS=99

# taille du contexte (équilibre entre mémoire et longueur de
contexte)
export OLLAMA_CONTEXT_LENGTH=32768
```

Étape 6 : les 5 commandes que vous utiliserez vraiment au quotidien

Ce sont les seules commandes à connaître. Tout le reste est optionnel.

```
# 1. démarrer Ollama (s'il ne tourne pas déjà en tant que service)
ollama serve

# 2. récupérer un nouveau modèle
ollama pull glm-4.7-flash

# 3. lister les modèles installés
ollama list

# 4. démarrer Claude Code avec un modèle précis
claude --model glm-4.7-flash

# 5. vérifier ce qui tourne et combien de mémoire est utilisée
ollama ps
```

Commandes bonus pour quand quelque chose casse :

```
# consulter les logs d'Ollama (utile pour le débogage)
journalctl -u ollama -f    # linux
tail -f ~/.ollama/logs/server.log    # macos

# supprimer un modèle pour libérer de l'espace disque
ollama rm glm-4.7-flash

# mettre à jour tous les modèles vers leur dernière version
ollama pull glm-4.7-flash
```

Étape 7 : configurer un repli vers le cloud (quand le local ne suffit pas)

Parfois, l'inférence locale n'est pas le bon outil. Longues sessions complexes exigeant la qualité d'un modèle de pointe. Déplacement avec une machine légère. Tâches où 25 tokens par seconde ne suffisent pas.

Pour ces moments, utilisez OpenRouter comme repli. Même configuration de variables d'environnement. Même expérience Claude Code. Le palier gratuit est généreux.

Configuration OpenRouter :

1. Créez un compte gratuit sur openrouter.ai.
2. Générez une clé d'API (le palier gratuit = 5 dollars de crédits à l'inscription, puis paiement à l'usage).
3. Définissez les variables d'environnement :

```
export ANTHROPIC_BASE_URL=https://openrouter.ai/api/v1
export ANTHROPIC_AUTH_TOKEN=your-openrouter-key
export ANTHROPIC_API_KEY=""
```

1. Choisissez votre modèle :

```
# modèles de pointe disponibles sur le palier gratuit d'OpenRouter
claude --model anthropic/claude-3.5-sonnet    # claude de secours
claude --model qwen/qwen3-coder-480b           # qwen massif
claude --model google/gemini-2.5-pro           # le meilleur de
google
```

Le flux de bascule :

```
# local par défaut (enregistré dans .zshrc)
export ANTHROPIC_BASE_URL=http://localhost:11434

# basculer vers le cloud pour du travail lourd (surcharge
temporaire)
ANTHROPIC_BASE_URL=https://openrouter.ai/api/v1
ANTHROPIC_AUTH_TOKEN=your-key claude --model qwen/qwen3-coder-480b

# revient automatiquement en local à la prochaine session
```

Étape 8 : configurer un CLAUDE.md pour les modèles locaux

CLAUDE.md est le fichier de contexte que Claude Code lit au début de chaque session. C'est là que vous décrivez votre projet, vos préférences et vos habitudes au modèle.

Les modèles locaux profitent davantage d'un bon CLAUDE.md que le modèle Claude complet. Les modèles locaux disposent de moins de données d'entraînement sur votre stack spécifique. Un fichier de contexte détaillé comble cet écart.

Le modèle de CLAUDE.md à utiliser :

Créez ceci à la racine de chaque projet :

```
# contexte du projet

## ce que c'est
[une phrase sur le projet]

## stack
- langage : [typescript / python / autre]
- framework : [next.js / fastapi / etc]
- base de données : [postgres / supabase / etc]
- dépendances clés : [listez les principales]

## comment le projet est structuré
[brève explication de l'arborescence]
/src : [ce qui vit ici]
/lib : [ce qui vit ici]
/components : [ce qui vit ici]

## mes préférences de code
- [par ex. toujours utiliser async/await, pas .then()]
- [par ex. préférer le fonctionnel au orienté classe]
- [par ex. le pattern de gestion d'erreur que j'utilise]
- [par ex. conventions de nommage]

## ce qu'il ne faut jamais faire
- [choses qui casseraient le projet]
- [patterns que je ne veux pas voir utilisés]

## contexte de la tâche en cours
[ce sur quoi je travaille maintenant, à mettre à jour à chaque
session]

## problèmes connus
[tout ce qui est cassé et que le modèle ne devrait pas corriger de
travers]
```

Pourquoi cela compte spécifiquement pour les modèles locaux :

Les modèles locaux n'ont pas la même compréhension implicite de votre base de code que Claude Sonnet, qui bénéficie d'un entraînement plus large. Un CLAUDE.md détaillé compense. Dix minutes à l'écrire. Des heures de correction évitées.

Dépannage

Problèmes réels rencontrés en conditions réelles. Corrections réelles qui ont fonctionné.

Claude Code se bloque au premier message

Cause : le modèle n'est pas chargé, ou Ollama ne tourne pas.

```
# vérifier si Ollama tourne
curl http://localhost:11434/api/tags

# s'il ne tourne pas, démarrez-le
ollama serve

# vérifier que le modèle est bien récupéré
ollama list
```

Erreurs 404 sur `/v1/messages`

Cause : version d'Ollama inférieure à 0.14.

```
# vérifier la version
ollama --version

# mettre à jour
brew upgrade ollama    # macos
curl -fsSL https://ollama.com/install.sh | sh    # linux
```

Réponses lentes (sous 5 tokens par seconde)

Cause : modèle qui tourne sur le CPU, ou modèle trop grand pour la vram.

```
# vérifier le processeur utilisé
ollama ps

# si CPU uniquement, deux options :
# a) utiliser un modèle plus petit / quantifié
ollama pull glm-4.7-flash    # le plus petit bon modèle

# b) vérifier la détection du GPU
ollama info
```

L'appel d'outils ne fonctionne pas / Claude Code ne peut pas éditer les fichiers

Cause : certains modèles ont une implémentation faible de l'appel d'outils.

Correction : basculez vers un modèle avec un support natif de l'appel d'outils.

Meilleur support d'appel d'outils, dans l'ordre :

1. glm-4.7-flash (natif, fiable) ;

2. qwen3-coder (bon, ratés occasionnels) ;
3. deepseek v4 (solide, lourd).

À éviter : les modèles llama pour l'usage agentique d'outils. Ils sont excellents en conversation, mais leur appel d'outils est peu fiable pour les flux Claude Code.

Le modèle est déchargé entre les requêtes

Cause : le keep-alive par défaut d'Ollama est de 5 minutes.

```
# régler à 1 heure
export OLLAMA_KEEP_ALIVE=1h

# ou sans expiration (garder chargé jusqu'au redémarrage)
export OLLAMA_KEEP_ALIVE=-1
```

Problème de repli sur `ANTHROPIC_API_KEY`

Si une vraie clé d'API Anthropic est définie quelque part dans votre environnement, Claude Code peut y retomber au lieu d'utiliser Ollama.

Correction :

```
# effacer explicitement
export ANTHROPIC_API_KEY=""

# vérifier qu'aucune autre variable d'environnement ne surcharge
env | grep ANTHROPIC
```

Les trois doivent correspondre :

```
ANTHROPIC_BASE_URL=http://localhost:11434
ANTHROPIC_AUTH_TOKEN=ollama
ANTHROPIC_API_KEY=
```

Le calcul du coût

Voilà pourquoi vous mettez tout cela en place.

niveau d’usage	coût API Anthropic / mois	coût Ollama / mois	économies
léger (1h/jour)	~40 $	0 $	40 $
moyen (3h/jour)	~120 $	0 $	120 $
intensif (6h/jour)	~280 $	0 $	280 $
abonnement Claude Max	100-200 $	0 $	100-200 $

Le retour sur investissement matériel :

Une configuration correcte qui fait bien tourner tout ceci :

- Mac mini M2 (16 Go) : 600 $;
- Nvidia RTX 4070 (12 Go de vram) : 550 $;
- montée en ram supplémentaire : 80-120 $.

Seuil de rentabilité pour une facture d’API de 120 $ par mois : 5 mois.

Après cela, chaque mois est de l’économie pure. Durablement.

Les limites honnêtes

Ce guide est franc sur les compromis. Les voici.

Les modèles locaux sont plus lents que Claude. L’infrastructure d’Anthropic tourne à 100-200 tokens par seconde. Votre machine tourne à 15-35. Pour les longues sorties, cela s’additionne. Ce n’est pas rédhibitoire. C’est juste un fait à connaître.

Les modèles locaux sont moins capables que Claude Sonnet 4.5. glm-4.7 est très bon. qwen3-coder est réellement impressionnant. Aucun des deux n’est Claude Opus. Pour la plupart des tâches de code, l’écart n’a pas d’importance. Pour des problèmes d’architecture réellement difficiles, utilisez le repli vers le cloud.

La compatibilité API Anthropic d’Ollama est encore en maturation. Les cas limites en streaming et en appel d’outils sont corrigés régulièrement. Consultez les notes de version d’Ollama avant les mises à jour majeures. Quand quelque chose casse après une mise à jour, vérifiez si les drapeaux d’appel d’outils ont changé.

Votre machine doit être allumée. Claude Code dans le cloud fonctionne depuis n’importe quel appareil. Ollama en local ne fonctionne que lorsque votre machine tourne. Si vous vous déplacez souvent avec un appareil léger, mettez en place le repli cloud via OpenRouter de l’étape 7.

Aide-mémoire de démarrage rapide

(Gardez ceci sous la main. Vous en aurez besoin quand vous mettrez ça en place à 23h et que vous ne vous souviendrez plus des commandes.)

```
# 1. installer Ollama
brew install ollama    # macos
curl -fsSL https://ollama.com/install.sh | sh    # linux

# 2. vérifier que la version est 0.14 et plus
ollama --version

# 3. récupérer votre modèle
ollama pull glm-4.7-flash

# 4. définir les variables d'environnement (à ajouter à .zshrc
pour les rendre permanentes)
export ANTHROPIC_BASE_URL=http://localhost:11434
export ANTHROPIC_AUTH_TOKEN=ollama
export ANTHROPIC_API_KEY=""

# 5. démarrer Ollama
ollama serve

# 6. lancer Claude Code
claude --model glm-4.7-flash

# 7. vérifier que ça marche
# tapez : sur quel modèle tournes-tu ?
# doit afficher : glm-4.7-flash ou le modèle que vous avez
récupéré
```

C’est fait. Aucune facture d’API. Même flux de travail. Mêmes résultats.

La suite

Si vous avez mis tout cela en place et que ça tourne, le prochain mouvement évident consiste à router intelligemment certains flux de travail entre le local et le cloud.

- Sessions agentiques lourdes sur des tâches simples → local (gratuit).
- Raisonnement complexe sur des problèmes difficiles → cloud (coût minimal).
- Bases de code sensibles → toujours en local.
- Tâches ponctuelles rapides → local.

Cette logique de routage est la couche suivante. Vous la construisez une fois, elle ne coûte presque rien ensuite.