

IT 資格 道 場

GitHub Copilot GH-300 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

Microsoft Learn「GH-300: GitHub Copilot 学習ガイド」Skills measured as of
August 7, 2026(2026-10-13 取得)準拠
2026年10月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

目次

この教材群の使い方(3点セット)

この試験が問うもの

第1章 Use GitHub Copilot responsibly [1.1.a/1.1.b/1.1.c/1.2.a/1.2.b]

- 幻覚(もっともらしい誤り)と知識の時点 [1.1.a]
- 学習データの偏りと言語・分野による品質の差 [1.1.a]
- 生成コードの安全性・ライセンスの懸念と過信 [1.1.a]
- コンテキスト窓と大きな PR の限界 [1.1.a]
- Microsoft の責任ある AI の 6 原則 [1.1.b]
- 透明性と説明責任 (AI の関与の明示・人が最終責任を持つ) [1.1.b]
- 責任の分担と利用者の責任 [1.1.b]
- 害の種類(偏り・機密の漏れ・安全でないコード・知的財産) [1.1.c]
- 害と軽減策の対応(レビュー・テスト・フィルター・除外・スキャン) [1.1.c]
- エージェントの害と範囲の絞り方 [1.1.c]
- 検証の手段(人のレビュー・テスト・静的解析・code scanning) [1.2.a]
- 提案を受け入れる前の確認点(正しさ・安全・ライセンス・規約) [1.2.a]
- エージェントの作業を確かめる [1.2.a]
- 個人の運用(文脈を与える・理解してから採用・機密を入れない) [1.2.b]
- 組織の運用(ポリシー・安全機能の有効化・研修・フィードバック) [1.2.b]
- アプリケーションカードと透明性の文書 [1.2.b]
- 取り違えやすい点のまとめ [1.2.b]

サンプルはここまでです

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

この試験が問うもの

GH-300 は、GitHub Copilot を業務で使う開発者や管理者に向けた、中級の試験です。コードを書く力ではなく、機能の名前と設定、プランによる違い、責任ある AI の考え方、データの扱いを、場面に合わせて選べるかを問います。

項目	内容
試験コード	GH-300
試験時間	100 分
合格ライン	700 点以上 (1000 点満点の換算点)
言語	英語・スペイン語・ポルトガル語 (ブラジル)・韓国語・日本語

分野ごとの出題比率は、幅で示される。

- 分野 2 GitHub Copilot 機能の使用: 25～30%
- 分野 1 責任を持って使用する: 15～20%
- 分野 3 データとアーキテクチャ、分野 4 プロンプトエンジニアリングとコンテンツ作成、分野 5 開発者の生産性の向上、分野 6 プライバシー・コンテンツの除外・セーフガード: 各 10～15%

このテキストは、この 6 分野を同じ順番で第 1 章から第 6 章まで並べています。出題範囲は 2026 年 8 月 7 日時点のスキルに基づく。試験で問われるのは多くが一般提供の機能で、よく使われるプレビュー機能の問題が出ることもある。

機能の名前、プランの構成、使用量や上限の数値は、GitHub が更新するたびに変わります。本文の数値は書いた時点の GitHub Docs の記述で、購入や契

約の判断には使えません。試験では「どのプランのどの機能か」という関係が問われるので、数値より関係を覚えておけば十分。

SAMPLE

第1章 Use GitHub Copilot responsibly

[1.1.a/1.1.b/1.1.c/1.2.a/1.2.b]

この分野の軸は一つで、Copilot が出すものは下書き、採用の責任は人にあります。リスクの名前と原因、Microsoft の 6 原則、害と対策の組み合わせ、採用前の検証、個人と組織の運用が、場面つきの問題として出ます。選択肢には「もっともらしい誤り」が混ざるので、名前と定義を取り違えない読み方を身につけます。

幻覚(もっともらしい誤り)と知識の時点 [1.1.a]

幻覚(hallucination)は、言語モデルが「もっともらしく聞こえるが、事実と違う、与えた文脈に支えがない、または作り上げた」出力を返す現象です。日常語の「見間違い」と違って、モデルは誤っていても自信ありげな文体を崩しません。構文が正しく動きそうに見えるので、読む側が誤りに気づきにくくなります。Copilot の応用カードは、幻覚を大規模言語モデルの既知のリスクとし、人のレビューが要る主な理由に挙げています。[^c1-1]

幻覚はコード補完に限りません。pull request の説明文を補う機能(pull request text completion)でも起きます。この機能は、タイトル、既存の説明文、コミットのタイトル、差分の一部、最近見た pull request と issue のタイトルから文章を続けます。そのため、実際には行っていない変更を書いたような文が出ることがあります。公開する前に必ず読み、事実と合うか確かめます。[^c1-1]

Copilot Chat にも同じ性質があります。Chat はコードに関する質問のために作られており、コード以外の話題への回答は、的外れだったり意味をなさなか

ったりします。最適化は主に英語で行われているので、英語以外のプロンプトでは性能が変わることがあります。[^c1-23]

知識の時点は別のリスクです。大規模言語モデル (LLM) は、学習が終わった時点までの情報で答えます。新しく出たライブラリの使い方や、最近になって非推奨とされた API の判断は、古いまま返ることがあります。提案が動いたとしても、今の推奨に沿っているとは限りません。

非決定的であることも押さえます。Copilot の出力は確率的なモデルが作るので、同じ入力から毎回同じ答えが返るとは限りません。反対に、似た入力には似た出力が返るので、別の利用者に似たコードが出ることもあります。Trust Center の FAQ は、無関係な 2 人が Java で新しいファイルに quicksort を書き始めれば、同じ出力になる可能性が高いと例示しています。GitHub は出力の所有権を主張しないと明記しています。[^c1-2]

現象	中身	向き合い方
幻覚	もっともらしいが事実と違う、または文脈に支えのない出力	人がレビューして事実と照らす
知識の時点	学習した時点までの知識で答える	新しい API と非推奨の判断は公式の文書で確かめる
非決定的	同じ入力でも出力が変わる。別の利用者に似た出力が出る	再現を前提にせず、毎回レビューとテストを通す

試験は原因と現象の組み合わせで問います。「存在しないメソッドを自信をもって提案した」は幻覚、「半年前に非推奨になった API を勧めた」は知識の時点です。「同じプロンプトで 2 回目は別のコードが出た」は非決定的です。この

3つを入れ替えた選択肢が並ぶので、名前と定義を結びつけておかなければ選べません。

pull request の説明を書いている、Copilot が「キャッシュを追加し、テストを更新しました」と続けてきた場面を考えます。実際の差分には、キャッシュの追加がありません。これは pull request text completion の幻覚です。公開する前に差分と照らして書き直すのが正しい対応で、補完をそのまま受け入れるのは誤りです。タイトルとコミットメッセージを明確にすれば関連性は上がりますが、内容の正しさを確かめる責任は書き手に残ります。[^c1-4]

学習データの偏りと言語・分野による品質の差 [1.1.a]

提案の質は、その言語の学習データの量と多様さで変わります。応用カードは、JavaScript が公開リポジトリに豊富で最も手厚く扱える言語の一つなのに対し、公開リポジトリに少ない言語は手助けが難しくなると書きます。[^c1-3]

これが限られた範囲 (Limited scope) です。大量のコードで学習していても、複雑なコード構造や珍しい言語は扱いきれないことがあります。もう一つの制約として、インライン提案は書いているコードの文脈からしか提案できません。設計や全体構成の問題には気づけません。コードとコード関連の出力を作る機能であり、自然言語の文章を作る機能ではない点も、カードは明記しています。[^c1-3]

偏りも同じ根から出ます。学習データの出どころには、偏りと誤りが含まれます。Copilot はそれを引き継ぐことがあり、特定の言語や書き方に寄った、最適でない提案や不完全な提案を出します。軽減策は、多様な観点でレビューすることで、1人の好みだけで採否を決めないことです。[^c1-3]

機能ごとの限界も出題されます.Chat のカードと inline suggestions のカードから、試験に出やすいものを表にします。[^c1-23][^c1-3]

機能	限界
PR の要約 (GitHub.com)	追加と削除の合計が 400 行を超えるファイルは要約から除かれる。自動では更新されない
コミットメッセージの生成	複雑な変更、短い diff、新しい言語が苦手。自動では更新されない
PR の文章補完	主に英語向け。巨大な PR では材料が API 呼び出しに入りきらず、提案が出ないか精度が落ちる
Spaces	サイズに上限があり、入れた内容の一部だけが使われる。意図を取り違えることがある
要約・メッセージの生成全般	PR やコミットに不適切な語があれば、生成結果にも同じ語が入りうる

要約やメッセージが自動で更新されないことは、場面問題で効きます。PR にコミットを追加した後に古い要約をそのまま公開するのは誤りで、再度生成を求めて新しい要約も同じ注意で読み直します。

ある会社の主力言語は、公開コードの少ない独自言語です。Copilot の提案の採用率が、JavaScript のチームより明らかに低いと分かりました。原因は学習データの量と多様さの差です。対策は、関連ファイルを開いて文脈を厚くし、提案をレビューとテストで確かめることです。Copilot の故障ではなく、プランの問題でもありません。上位のプランにしても学習データは増えません。

生成コードの安全性・ライセンスの懸念と過信 [1.1.a]

構文が正しく見えても、安全とは限りません。Copilot は、周りのコードの文脈から提案を作ります。そのため、注意しないと機密情報や脆弱性を持ち込みます。応用カードは、安全が要る用途では、生成コードを必ずレビューしテストするように求めます。[^c1-3]

典型の例が 2 つあります。1 つ目はハードコードされたパスワード、2 つ目は SQL インジェクションです。次のコードは、同じ検索を文字列の連結と、プレースホルダーで書いたものです。実際に実行すると、連結の方は全件の 2 行が返り、プレースホルダーの方は 0 行でした。

```
import sqlite3

c = sqlite3.connect(":memory:")

c.execute("create table users(name text, pw text)")

c.execute("insert into users values('alice','s3cret'),
('bob','hunter2')")

name = "x' OR '1'='1"

bad = c.execute("select * from users where name='%s'" %
name).fetchall() # 2 行 (全件)

good = c.execute("select * from users where name=?",
(name,)).fetchall() # 0 行
```

連結の方は、入力が SQL の一部として解釈され、条件が常に真になります。Copilot がこの形を提案しても、構文は正しく動きますが、動くことと安全であることは別物です。パスワードの方は、次のように環境変数から読む形へ直します。環境変数が未設定なら `KeyError` で止まり、設定漏れも見逃しません。

```
import os  
  
pw = os.environ["DB_PASSWORD"] # 未設定なら KeyError
```

過信 (overreliance) は、誤りや不完全な出力を、誤りに気づきにくいという理由で受け入れてしまうことです。応用カードは、結果として生産性の低下、信頼の喪失、金銭的な損失、心理的・身体的な害などが起こりうると書きます。例に挙がるのは、医師が誤った AI 出力を受け入れる場面です。[^c1-4]

ライセンスの懸念は、提案がまれに学習データの公開コードと一致することから来ます。確率は低くても、生成が確率的なので一致は出ます。応用カードの注意書きは明快です。生成されたコードに伴うセキュリティ上の脆弱性、バグ、知的財産の侵害のリスクは、利用者が負います。[^c1-4]

Copilot の強みと弱みも、ここで分けておきます。得意なのは、テストと繰り返しの多いコードの作成、デバッグと構文の修正、コードの説明とコメント付け、正規表現の生成です。想定していないのは、コーディングと技術に関係しない依頼への応答と、人の専門性や技能の代わりになることです。個人データを扱う判断や、設計の全体判断は、人が持ちます。[^c1-5]

コンテキスト窓と大きな PR の限界 [1.1.a]

Copilot が一度に読める範囲には上限があり、これがコンテキスト窓です。Microsoft の入門教材は、Copilot の窓を概ね 200～500 行、または数千トークンと書きます。同じ教材は、Copilot Chat が 4k トークンの窓で動くとも書きます。教材自身も、この値は Copilot の実装と版によって変わりうると書いており、今の製品の窓の値は一次情報で確かめられないため、ここでは教材の値を挙げます。[^c1-6]

窓に上限があるので、使い方の工夫が2つ出ます。複雑な問題は、小さく焦点の絞られた質問に分けます。関連するコードの断片を、プロンプトに添えます。

窓の限界は pull request にも現れます。とても大きな pull request では、補完に必要な内容が API 呼び出しに入りきりません。すると、期待した提案が出ない、または文脈とのずれが大きくなります。大きな変更は、レビューできる単位に分けて出すのが近道です。[^c1-3]

Microsoft の責任ある AI の 6 原則 [1.1.b]

GitHub は、Microsoft の 6 つの中核原則に従っています。この原則は Responsible AI Standard に組み込まれ、AI アプリケーションの設計、構築、テストの指針になっています。[^c1-7]

6 つの違いは、Microsoft が各原則に付けた問いで見分けます。[^c1-8]

原則	一言で	場面の見分け方
公平性 (fairness)	すべての人を公平に扱う	言語や地域、属性で提案の質や扱いが偏らないか
信頼性と安全性 (reliability and safety)	想定どおり安全に動く	想定外の使い方や状況でも害を出さずに動くか
プライバシーとセキュリティ (privacy and security)	個人データを守り、攻撃に耐える	個人情報が出入りしないか、システムが保護されているか
包括性 (inclusiveness)	背景や能力が違う人を含める	障害の有無などに関わらず使えるか
透明性 (transparency)	理解できるようにする	能力と限界を、利用者が正しく分かる形で示しているか
説明責任 (accountability)	人が責任を負う	人が監督し、制御できる体制があるか

選択肢には、6 つに入らない語が混ざります。効率性、収益性、創造性、持続可能性は、この 6 つに入っていません。信頼性と安全性は 1 つの原則で、プライバシーとセキュリティも 1 つの原則です。語が 2 つ並んでいても、原則を 2 つとは数えません。

公平性と包括性は間違いやすい組で、公平性は結果の扱いが人によって偏らないこと、包括性は能力や背景の違う人が使える設計であることを指します。同じく、プライバシーとセキュリティは「守る」、説明責任は「責任を負う」、透明性は「理解できるようにする」と、動詞で区別します。

場面	当てはまる原則
障害のある開発者も Copilot を使えるように、操作の方法を増やした	包括性
提案が AI によるものだと分かるように、画面で区別して示す	透明性
提案を採用する前に、人がレビューする	説明責任
想定外の入力でも有害な出力を出さないよう、テストを重ねる	信頼性と安全性
プロンプトから個人データを除く	プライバシーとセキュリティ
言語によって提案の質が極端に変わらないよう評価する	公平性

透明性と説明責任 (AI の関与の明示・人が最終責任を持つ) [1.1.b]

透明性は、AI の仕組み、限界、使い方を、利用者が分かる形で示すことで、GitHub の応用カードはその具体的な文書です。能力、想定する用途、限界を示して、原則を運用に落とし込みます。[^c1-7]

画面上の作りも、透明性と人の監督を支えます。インライン提案は、周りの内容と見た目で区別され、利用者が明示的に受け入れたときだけファイルに反映されます。受け入れる、却下する、無視して打ち続ける、のどれも選べます。利用者の操作なしにコードが変わることはありません。人が判断の輪の中にいる設計 (human-in-the-loop) の例です。[^c1-1]

クラウドエージェントでも、AI の関与を追えるようにしてあります。エージェントのコミットは Copilot が作者になり、タスクを始めた人が共同作者に記録されます。コミットには署名が付き、GitHub 上で「Verified」と表示されます。コミットメッセージにはセッションログへのリンクが入るので、誰が始め、何が作り、なぜ変えたのかを、レビューや後の監査で追えます。[^c1-9]

説明責任は、AI の出力を使った結果の責任を、人（組織）が持つことです。AI は責任を負えません。応用カードも、Copilot を道具として使い、人の代わりにしないよう求めます。提案を受け入れる前に、正確さと妥当さを確かめるのは利用者の責任です。[^c1-4]

透明性と説明責任は混同されがちです。

原則	側	例
透明性	何が起きているかを分かるようにする	AI の関与の明示、限界の公開、応用カード
説明責任	誰が責任を負うかを定める	人のレビュー、承認、最終判断

選択肢が「AI の関与を明示する」なら透明性、「人が最終的に承認する」なら説明責任と読めば足ります。

責任の分担と利用者の責任 [1.1.b]

責任ある AI は、GitHub と顧客の共同の約束です。GitHub は安全、公平、透明を中心に AI アプリケーションを作り、顧客は自分の文脈で責任ある導入と利用をする役を担います。GitHub が安全機能を持つから顧客は何もしなくてよい、という選択肢は成り立ちません。反対に、GitHub に責任がないという選択肢も正しくありません。[^c1-4]

担い手	役割の例
GitHub	安全、公平、透明を中心にした設計。コンテンツフィルター、公開コードの一致の検出、敵対的テスト、応用カード
顧客と利用者	自分の文脈での責任ある導入と利用。提案のレビューと検証、ポリシーの設定、研修、機密を入れない習慣

利用者の責任は、はっきり書かれています。提案を受け入れる前に、正確さと妥当さ確かめることです。受け入れた後も、要件を満たし、誤りやセキュリティ上の問題がないことを検証します。応用カードの注意書きは、生成されたコードに伴う脆弱性、バグ、知的財産の侵害のリスクは、利用者が負うとしています。[^c1-1][^c1-4]

導入する側への注意も、4 つあります。

- **重大な判断と敏感な領域:** 教育、雇用、金融、公的給付、医療、住宅、保険、法務などへの法的または重大な影響がある判断では、結果を評価する。影響を受ける人が、判断の仕組みを理解し、異議を申し立て、入力データを直せるようにする
- **法的・規制上の考慮:** 業種や場面によって、AI の利用が適切でないことがある。適用される法的義務を評価する

- **コードの話題に絞る:** インライン提案は、コードとコード関連の提案を作るための機能。エディタの中身をそれに絞ると、質が上がる
- **最新に保つ:** Copilot は進化中の技術。VS Code、Visual Studio、JetBrains の IDE では、拡張機能の自動更新が既定で有効

エージェントを敏感な領域で設計する場合は、さらに慎重さが要ります。エージェントの行動が元に戻せない、または影響が大きい領域では、追加の予防策を取ります。[^c1-4][^c1-10]

保険会社が、契約の審査の判断を Copilot の提案に任せたいと言っている場面です。審査は保険という敏感な領域での重大な判断なので、提案をそのまま採用してはいけません。結果を評価し、影響を受ける人が判断の仕組みを理解して異議を申し立てられる形にします。人が責任を負う体制を残します。

害の種類(偏り・機密の漏れ・安全でないコード・知的財産) [1.1.c]

害は、偏り、機密の漏れ、安全でないコード、知的財産の 4 種類に分けて覚えます。それぞれ、原因と、利用者に出る形が違います。

- **偏り:** 学習データの偏りが提案に出る。特定の言語や書き方への寄りになる
- **機密の漏れ:** 文脈にあった機密や脆弱性を、提案が持ち込む。プロンプトに入れた機密も含む
- **安全でないコード:** 脆弱性を含む提案。ハードコードの秘密、SQL インジェクションなど
- **知的財産:** 提案が公開コードと一致する。ライセンスの義務が生じうる

知的財産の害は、公開コードとの一致から生まれます。Trust Center の FAQ は、出力が公開コードと一致した場合は、組織のオープンソースの方針に沿っ

て扱うよう求めます。一致を見つけたら、使うかどうかを決める前に、利用できる情報を調べるのが妥当な手順だとしています。ライセンスによっては、帰属表示やソースの開示といった義務が生じます。[^{^c1-2}]

偏りと安全でないコードはモデルの制約から、機密の漏れは文脈の与え方と環境の設定から来ます。害の原因が違えば、効く対策も違います。エージェント機能は、利用者の代わりにコマンドを実行しファイルを変更するので、害が一段大きくなります。応用カードは、CLI について、ファイル削除やディスクの初期化のような破壊的なコマンドに触れうると書きます。実行されたコマンドの責任を、AI に負わせることはできません。Chat のカードも、Windows の端末で破壊的なコマンドを提案する危険を挙げています。[^{^c1-10}][^{^c1-23}]

害と軽減策の対応(レビュー・テスト・フィルター・除外・スキャン)
[1.1.c]

害ごとに効く対策が決まっているので、表で対応を押さえます。

害	効く軽減策	補足
偏り	多様な観点でのレビュー	1 人の判断に寄せない
安全でないコード	人のレビュー、テスト、code scanning、コードの安全フィルター	フィルターがあっても、利用者のレビューは要る
機密の漏れ	コンテンツ除外、機密を入れない習慣	除外は Business と Enterprise の機能
知的財産	公開コード一致のフィルター (Block)、または参照表示 (Allow) での確認	Block でも、クラウドエージェントと CLI では一致が出る
有害な内容	コンテンツフィルター、入口のフィルター	プロンプトと応答の両方に効く

コンテンツフィルター (Content filtering) は、プロンプトと応答を調べ、有害、不快、安全でない内容を、利用者に出す前に止める安全の仕組みです。生成された提案は、コンテンツフィルターを通ります。Microsoft の入門教材は、入口の側でヘイトスピーチなどの不適切な内容と、個人データ(名前、住所、識別番号)を除くと説明します。GitHub 所有の Azure テナント上のプロキシは、プロンプトを操作する試みも通しません。インライン提案には、このほかに有害な出力を減らし誤用を防ぐコードの安全フィルターもあります。ただし、利用者のレビューは省けません。[^c1-1][^c1-11]

公開コードの一致 (Public code matching) は、提案が公開コードと一致するかを調べる機能で、設定が 2 つあります。

- **Block:** コードと周りの約 150 文字を、GitHub 上の公開コードと照合する。一致または近い一致があれば、提案を表示しない
- **Allow:** 一致する提案も表示する。一致の詳細と、元のリポジトリへのリンクを確認できる

Trust Center の FAQ は、この重複検出フィルターを、65 レキシム (平均で約 150 文字) 以上の一致を抑えるものと説明します。管理者が有効にします。[^c1-2][^c1-12]

Block でも、一致が完全になくなるわけではありません。クラウドエージェントと CLI は、設定が Block でも、公開コードと一致または近い一致のコードを生成することがあります。クラウドエージェントでは、一致がセッションログに表示され、詳細へのリンクが付きます。Chat は IDE で動作が違い、VS Code で Allow にしていると、一致のコードとライセンスを示すリンクが表示されます。Visual Studio、JetBrains、Eclipse、GitHub Mobile の Chat は、一致を遮断するフィルターを使います。どの設定でも、公開コードを使う時と同じ注意

として、厳密なテスト、IP スキャン、脆弱性の確認を省けません。[^c1-10][^c1-23]

コンテンツ除外は、指定したファイルを Copilot の文脈から外します。除外されたファイルでは、インライン提案が出ません。そのファイルの中身は、ほかのファイルの提案にも、Copilot の応答にも使われません。Copilot code review の対象からも外れます。機能は Copilot Business と Enterprise で使えます。設定できるのは、リポジトリ管理者、組織のオーナー、Enterprise のオーナーです。[^c1-13]

除外には限界があります。IDE が間接的に渡す情報（型情報、シンボルのホバ一定義、ビルド構成など）から、除外ファイルの意味情報が使われる可能性があります。シンボリックリンクと、リモートのファイルシステム上のリポジトリには効きません。VS Code などの Chat の Edit モードと Agent モードも、現時点では対象外です。除外を設定したから漏れが必ずゼロになる、という選択肢は誤りです。[^c1-13]

敵対的テストは、害の予防側の手段です。基盤モデルを更新した時や、学習データを大きく変えた時、モデルに有害な出力、安全でない出力、ポリシー違反の出力を引き出そうとする入力で試します。対象となるリスクには、有害な内容、知的財産のリスク、安全でないコードの生成が含まれ、結果を既存の基準と比べて悪化があれば人が確かめます。利用者が行う作業ではなく、GitHub がモデルを出す側で行う安全評価です。[^c1-1]

同じ「守る」でも、手段によって効く場所が違います。コンテンツ除外は入力側で Copilot が読む文脈を減らし、公開コードの一致のフィルターは出力側で提案が公開コードと一致するかを調べます。code scanning やテストは、提案を採用した後のコードを調べます。効く場所を取り違えた選択肢は、正しく見えても成り立ちません。

手段	効く場所	防ぐもの	防ぎきれないもの
コンテンツ除外	入力 (Copilot が読む文脈)	機密ファイルの内容が文脈に入ること	提案の脆弱性、公開コードとの一致
公開コードの一致 (Block)	出力 (提案)	公開コードと一致する提案の表示	機密の文脈への混入、脆弱性
コンテンツフィルター	入力と出力	有害、不快、安全でない内容	幻覚、偏り
レビュー、テスト、code scanning	提案の採用後	脆弱性、不具合	公開コードとのライセンス上の一致 (IP スキャンが別に要る)

金融系の組織が、API キーが書かれた設定ファイルを Copilot の文脈に出したくないと言っている場面です。効くのはコンテンツ除外です。公開コードの Block では、このファイルは守れません。Block は、提案が公開コードと一致するのを抑える機能で、機密の文脈を外す機能ではないからです。逆に、GPL のコードとの一致を防ぎたい場合は、コンテンツ除外を選んでも防げません。公開コードの一致の設定を使います。安全でない提案そのものを減らすには、人のレビューとテスト、code scanning が要ります。

エージェントの害と範囲の絞り方 [1.1.c]

エージェントは、コードを書くだけでなく実行して変更するので、害を防ぐ仕組みが入口から出口まで複数あります。

クラウドエージェントの安全策は次のとおりです。[^c1-9]

安全策	中身
権限昇格の制御	書き込み権限のある利用者の操作にだけ反応する。コメントや issue に仕込まれた隠し文字を除く
権限の制約	対象のリポジトリだけにアクセスする。書き込めるのは 1 つのブランチ (<code>@copilot</code> なら既存の PR のブランチ、それ以外は新しい <code>copilot/</code> ブランチ)。既定のブランチには直接書けない
ワークフローの承認	エージェントが出した PR で起動する Actions は、書き込み権限のある人が承認するまで動かない
シークレット	Actions の組織・リポジトリのシークレットは渡らない。 <code>copilot</code> 環境に入れたものだけが渡る
ファイアウォール	既定で有効。コードや機密データの持ち出しを、偶発でも悪意ある入力でも防ぐ
脆弱性の自動検出	生成したコードを CodeQL、シークレットスキャン、依存関係の分析で調べ、直そうとする
追跡できること	作者は Copilot、共同作者は起動した人。署名つきで Verified。コミットにセッションログへのリンク

CLI は、既定で起動したディレクトリとその下のファイルにだけ触れます。外のファイルに触れたい時と、ファイルを変更する前、危険になりうるコマンドを実行する前には、許可を求めます。許可の出し方には次の 3 つがあります。

- `--allow-tool` : 特定のツールだけを、確認なしで許可する

- `--allow-all-tools` : すべてのツールを、確認なしで許可する
- `--allow-all` : 全権限を許可する(対話の中では `/allow-all`)

自律的に動かす autopilot では、利用者が全権限を与えるのが普通です。禁じる指定 (`--deny-tool`) は許可より優先されます。[^c1-10][^c1-18]

CLI の範囲を絞る手段は、信頼するディレクトリ、承認の運用、サンドボックスです。サンドボックスには、手元のマシンへのアクセスを制限するローカルのもの、完全に隔離されたクラウドのものがあります。仮想マシンやコンテナなど、権限とネットワークを厳しく管理した環境で動かす方法もあります。[^c1-18]

computer use は、デスクトップのアプリを利用者の代わりに操作する、任意の機能です。同梱のプラグインは既定で無効で、使うには有効にします。Enterprise の管理者は、管理設定で無効にでき、利用者はそれを上書きできません。画面の見え方に頼るため、間違った操作や、意図しない操作をすることがあります。アプリに「Always allow」を付けると、次のセッションでもアプリ単位の確認なしに操作できます。機密情報を扱うアプリや影響の大きい操作があるアプリには付けず、不要になった承認は消します。途中で止めたい時は、CLI では Esc を 2 回押します。[^c1-9][^c1-10]

検証の手段(人のレビュー・テスト・静的解析・code scanning)

[1.2.a]

AI の出力を検証する必要があるのは、構文が正しくても安全や正確さを保証できず、幻覚、偏り、過信が起きるからです。検証は、疑う作業ではなく、道具の限界を前提にした手順です。手段は、人のコードレビュー、テストの実行、静的解析、セキュリティスキャン (code scanning) の 4 つです。best practices は、Copilot の作業を確かめる手順として、次の 3 つを挙げます。提案を採用する前に理解すること。機能と安全性だけでなく、読みやすさと保守しやすさも含

めて、提案を丁寧にレビューすること。自動テストと、linting、code scanning、IP スキャンなどの道具で、安全性と正確さの確認を自動化することです。
[^c1-5]

提案を理解できない時は、Copilot Chat にそのコードを説明させます。分からないまま採用しないための手段です。

生成されたテストも検証の対象です。Copilot はテストケースを提案でき、入力、期待値、アサーションを含めます。しかし、すべてのシナリオを網羅するとは限りません。テストが通っても、コードが正しいとは言い切れません。テスト自体もレビューします。[^c1-1]

Copilot code review には位置づけがあります。コメントを残す機能で、既定ではレビューの種類が「Comment」です。「Approve」でも「Request changes」でもありません。そのため既定では、必須の承認の数に入りません。Copilot が承認できる設定は、既定では無効です。有効にするなら、Enterprise、組織、リポジトリの各レベルで設定します。承認が有効でも、その後新しいコミットが push されると承認は取り消されます。[^c1-14]

Copilot code review は、人のレビューを補うもので、代わりにはなりません。応用カードは、大きく複雑な変更でコード品質の問題を見逃すことがあると挙げます。存在しない問題を指摘する誤検知があります。出てくる修正案が、不正確または安全でないこともあります。コメントは確かめずに反映してはいけません。custom instructions を設定すると、書き方の規約に沿ったレビューになります。[^c1-10][^c1-14]

Copilot が生成したテスト 10 本がすべて通った場面を考えます。通ったことは、テストが書かれた範囲で動いた証拠にすぎません。生成されたテストは、すべてのシナリオを網羅するとは限りません。境界の値、異常な入力、権限の

違いが抜けていないかをレビューし、足りなければ人が追加します。テストが通ったから検証が終わった、という選択肢は誤りです。[^c1-1]

Copilot code review が、存在しない問題を指摘した場面もあります。誤検知は、応用カードが挙げる既知の限界です。コメントを読み、コードと照らして、問題がなければ反映しません。親指を下げるボタンでフィードバックを返すと、改善につながります。指摘が正しいかどうかを決めるのは、人のレビューです。[^c1-10]

クラウドエージェントのセッションログは、作業内容と使ったツールを示します。ただし、利用者自身のレビューとテストの代わりにはなりません。[^c1-15]

提案を受け入れる前の確認点 (正しさ・安全・ライセンス・規約)
[1.2.a]

採用前の確認は、正しさ (意図どおりか)、安全 (脆弱性と秘密)、ライセンス (公開コードとの一致)、規約 (チームの書き方) の 4 点です。

確認点	見ること	使う手段
正しさ	意図どおりに動くか、境界の条件を満たすか	テストの実行、人のレビュー
安全	脆弱性、ハードコードの秘密がないか	code scanning、静的解析、レビュー
ライセンス	公開コードと一致していないか	参照表示の確認、IP スキャン
規約	チームの書き方、設計の流儀に合うか	レビュー、linting

公開コードの参照は、設定が Allow の時、一致した提案に付きます。IDE のログに、一致したコードの URL と、ライセンス (見つからなければ

NOASSERTION) が載ります。JetBrains の例では、ログで「Public Code References」を探します。URL とライセンスを確かめるまでは、使ってはけません。ライセンスの条件が合わなければ、別の実装に替えます。[^c1-16]

確認を省く理由にならない言い訳は、「テストが通った」「Copilot が出した」「構文が正しい」の 3 つです。テストは網羅しているとは限らず、Copilot は安全を保証せず、構文の正しさは安全とは別です。選択肢にこの言い訳が並んだら、採用前の確認が省かれているかを探します。

設定が Allow の組織で、提案に参照が付き、ライセンスが GPL-3.0 と表示されました。使うかどうかを決める前に、参照の URL で元のコードを見て、自社製品へ入れてよいライセンスかを確かめ、条件が合わなければ別の実装に替えます。Block に切り替えれば以後は一致する提案が表示されなくなりますが、すでに受け入れたコードは元には戻りません。[^c1-16][^c1-12]

Copilot が、決済処理のモジュールに対して、動くコードを提案しました。レビューしたところ、データベースのパスワードが文字列として書かれていました。提案を採用するなら、パスワードをそのままりポジトリに置けません。環境変数やシークレット管理に移し、リポジトリに入れない形に直します。テストが通ったことも、Copilot が出したことも、採用してよい理由になりません。レビューを Copilot code review だけに任せて人の承認を省くのも、既定では承認に数えられないため誤りです。

エージェントの作業を確かめる [1.2.a]

エージェントモードは、複数のステップのタスクを自律的に進め、ファイルを選び、変更を提案し、必要に応じて端末のコマンドも実行し、問題が出れば直しながら続けます。人は変更をレビューし、既定ではコマンドの実行も承認します。管理者や自分のエディタの設定で一部のコマンドを自動実行にもできますが、自動実行にするほど変更のレビューを省けません。[^c1-17]

CLI は、ファイルを変更するツールや、コマンドを実行するツールを初めて使う時に、承認を求めます。承認の選択肢は 3 つです。今回だけ許可、そのセッションの間は許可、拒否して別の指示を出す、です。提案されたコマンドは、承認する前に必ず注意して読みます。[^c1-18]

クラウドエージェントが pull request を作った場合も、手順は同じです。CodeQL、シークレットスキャン、依存関係の分析が自動で走りますが、人のレビューの代わりにはなりません。応用カードも、マージの前に、出力を十分にレビューするよう求めます。変更を読み、テストを走らせ、IP スキャンや脆弱性の確認を行い、それから承認します。[^c1-9][^c1-10]

エージェントに渡す課題の書き方でも、結果の確かめやすさが変わります。クラウドエージェントでも CLI でも、課題は範囲を絞ります。理想の issue は、問題の説明、完全な受け入れ基準、変更が要るファイルの手がかりを含みます。custom instructions でプロジェクトの作り方、テスト、検証の方法を伝えると、エージェントの成果物の質も上がります。[^c1-10]

チームが、クラウドエージェントに issue を割り当て、返ってきた PR を、CI が通っているからという理由でそのままマージしようとしている場面です。CI が通ることは必要条件の一つに過ぎません。セッションログで作業内容を追い、差分を読み、必要なテストを足してからマージします。エージェントの変更は `copilot/` ブランチに入り、既定のブランチへ直接は書かれません。

個人の運用(文脈を与える・理解してから採用・機密を入れない)

[1.2.b]

個人が責任を持って運用する習慣は、良い文脈を与えること、提案を理解してから採用すること、機密を入れないこと、フィードバックを返すことの 4 つです。

良い文脈は、質を上げる最も手軽な手段で、特別な設定を要しません。IDE では、関係のあるファイルを開き、関係のないファイルを閉じます。コメントに意図を書くと提案の向きが決まり、たとえば「再帰を使う」「シングルトンパターンを使う」と書けば、提案されるアルゴリズムの種類が変わります。Copilot Chat では、役に立たなくなった過去のやり取りを会話から消すか、新しい会話を始めます。IDE の Chat では、キーワードで特定のタスクや文脈に集中させます。[^c1-5][^c1-1]

安全なコーディングの習慣も、運用の一部です。ハードコードしたパスワードと SQL インジェクションを避け、レビューの習慣を保ちます。応用カードは、Copilot が構文の正しいコードを作れても、安全とは限らないとして、この習慣を求めます。エディタの中身をコードとコードに関する情報に絞って、Copilot をコードの話題に集中させると、提案の質が上がります。[^c1-4]

機密情報を、プロンプトや開いているファイルに入れないことも、個人の運用に入ります。Copilot はエディタの文脈を材料にするので、開いているファイルが入力になります。パスワード、秘密鍵、個人データは、プロンプトにも文脈にも入れません。Chat が Web 検索を使う設定では、プロンプトの内容と追加の文脈から検索クエリが作られるため、機密を書かない習慣がいっそう要ります。[^c1-23]

秘密鍵を含むファイルを開いたまま Copilot Chat に質問した場面を考えます。開いているファイルは文脈として使われうるので、鍵の内容が入力に含まれます。個人にできる対応は、関係のないファイルを閉じることと、機密をリポジトリやプロンプトに置かない習慣です。組織として防ぐなら、そのファイルをコンテンツ除外の対象にします。Block は、このリスクには効きません。[^c1-5][^c1-13]

フィードバックは、品質改善に返す手段です。インライン提案は受け入れるか却下するかがそのままフィードバックになり、Copilot Chat の個々の応答には親指を上げる、下げるのアイコンがあります。VS Code の Copilot Chat 拡張についての報告は、`microsoft/vscode-copilot-release` リポジトリに issue で出します (Visual Studio は Send feedback ボタン、JetBrains は share feedback のリンク)。不適切な内容を見つけたら、copilot-safety@github.com へ報告します。[^c1-5][^c1-19][^c1-2]

道具の選択も責任の一部で、best practices は補完とチャットの使い分けを示します。インライン提案は、コード片、変数名、関数の補完、繰り返しの多いコード、コメントからの生成、テスト駆動開発のテスト生成に向きます。Copilot Chat は、コードについての質問、大きなコードの生成と反復、ペルソナを決めた依頼に向きます。複数ステップの作業はエージェントの出番です。用途に合わない道具では、質が上がりません。[^c1-5]

CLI を運用するなら、起動する場所に注意します。信頼できるディレクトリからだけ起動します。ホームディレクトリからは起動しません。実行できるファイルを信頼できないディレクトリで使うのは危険です。機密なデータのあるディレクトリで起動すると、意図せずファイルをさらすおそれがあります。権限の範囲づけは経験則による仕組みで、信頼するディレクトリの外のファイルがすべて守られる保証はありません。[^c1-18]

`--allow-all-tools` のような自動承認のオプションは、特に注意が要ります。Copilot は、あなたと同じ権限でファイルにアクセスし、あなたが実行できるシェルコマンドを事前の承認なしに実行できるので、データの損失や破損の危険が増えます。特定のツールだけを許すなら `--allow-tool`、特定のツールを禁じるなら `--deny-tool` を使います。`--deny-tool` は、`--allow-all-tools` と `--allow-tool` より優先されます。[^c1-18]

ある開発者が、ホームディレクトリで Copilot CLI を起動し、作業を速くするために `--allow-all-tools` を付けました。問題は、起動場所が信頼するディレクトリではない点と、承認なしで全ツールを許した点の 2 つです。直すには、対象のプロジェクトのディレクトリで起動します。承認は必要なツールだけに絞り、危険なツールは `--deny-tool` で禁じ、サンドボックスで動かして範囲を絞る方法もあります。

組織の運用(ポリシー・安全機能の有効化・研修・フィードバック)

[1.2.b]

組織の運用は、管理者が設定する安全機能（公開コードの一致の Block、コンテンツ除外、ポリシーによる機能の絞り込み）と、利用者への周知と研修に分かれます。ポリシーの例として、エージェントモードを IDE で無効にでき、無効にすると利用者のモード選択に Agent が表示されません。[^c1-20][^c1-17]

公開コードの一致の設定は、組織や Enterprise から継承されます。Enterprise Cloud の組織で Copilot のシートを割り当てられた人は、個人のアカウントの設定でこの項目を変えられません。組織や Enterprise の設定が継承されます。個人の設定画面では、「Suggestions matching public code」のプルダウンで Allow か Block を選びます。[^c1-12]

項目	個人	組織・Enterprise
公開コードの一致	個人のアカウントの設定で Allow か Block を選ぶ。組織のシートなら継承される	組織と Enterprise が設定し、メンバーに継承される

項目	個人	組織・Enterprise
コンテンツ除外	個人のプランの機能としては書かれていない (Business・Enterpriseの機能)。リポジトリ単位では、リポジトリ管理者が自分のリポジトリで設定する	組織のオーナーとEnterpriseのオーナーが設定する
フィードバック収集とプレビュー	組織が opt in した時に、利用者に選択肢が出る	管理者が「Copilot in GitHub.com」を有効にして opt in する

Copilot Business か Enterprise のプランで「Copilot in GitHub.com」を有効にすると、2 つの追加の選択肢が表示されます。

- **Opt in to user feedback collection:** 有効にすると、利用者が選ばれた Copilot の機能にフィードバックを送れる選択肢が出る
- **Opt in to preview features:** 有効にすると、一般提供前の新機能を試せる

プレビュー機能は、欠陥を含みうることを前提にします。機能が変更されたり中止されたりする可能性もあります。試験にも、よく使われるプレビュー機能の問題が出ることがあります。[^c1-22]運用では、本番の作業をプレビュー機能に依存させない判断が、責任ある使い方です。[^c1-20]

研修は運用を支える柱で、利用者が Copilot の限界 (幻覚、偏り、窓の上限)、提案の検証手順、機密を入れない習慣、エージェントの承認の扱いを知っているかが組織の安全を左右します。管理者が機能をどれだけ絞っても、利用者の理解が足りなければ、レビューは形だけになりかねません。公開コードと一致した出力はその方針に沿って扱うため、組織のオープンソースの方針と AI を使う際の規約も利用者に周知します。[^c1-2]

フィードバックの経路も運用に入れます。組織で利用者のフィードバックを集めたいなら、管理者が上の opt-in を有効にします。個々の利用者は、親指のアイコンで応答を評価します。見つけた問題は、報告先に伝えます。

ある企業が Copilot Business を導入しました。要件は 3 つです。公開コードと一致する提案は出たくない。契約関連のリポジトリの内容は Copilot に読ませたくない。社員が Copilot のプレビュー機能を勝手に試さないようにしたい。対応は、公開コードの一致を Block にすること、契約関連のリポジトリにコンテンツ除外を設定すること、「Opt in to preview features」を無効のままにすることです。コンテンツ除外が使えるのは Business と Enterprise なので、プランの条件も満たしており、追加の契約は要りません。

アプリケーションカードと透明性の文書 [1.2.b]

アプリケーションカードは、AI アプリケーションの仕組み、アプリケーションの所有者が選べる事項が性能と動作に与える影響、そして限界を示す GitHub の文書です。カードは、技術だけを見ず、人と環境を含めた全体で考える大切さも示します。GitHub のカードは、Copilot Chat、インライン提案、エージェント機能の 3 種類があります。プラットフォームカードは、AI のプラットフォームサービス向けです。[^c1-7][^c1-21]

カードの目的は、開発や導入の参考になることと、影響を受ける利用者や関係者に共有できることです。透明性の原則を運用に落とす文書です。

インライン提案と Chat のカードは、同じ 11 の節で構成されます。

1. 概要
2. 主要な用語
3. 主な機能
4. 想定する用途

5. モデルと学習データ
6. 性能
7. 限界
8. 評価
9. 安全の構成要素と軽減策
10. 展開と採用のベストプラクティス
11. さらに学ぶ

覚えるのは、節の名前そのものより、どの情報がカードに載るかです。押さえておく情報は次の3つです。

- **想定する用途:** テスト生成、コメントからのコード生成など
- **限界:** 限られた範囲、偏り、セキュリティのリスク、公開コードとの一致、不正確なコード
- **安全の構成要素:** 入出力の処理、コンテンツとコードの安全フィルター、公開コードの一致、人の監督

カードを読む場面は導入の判断の時で、たとえば導入前に「この機能は何が苦手か」を知りたいなら、限界の節を読みます。自分のチームに合うかを判断し、責任ある運用を設計する材料になります。カードは製品の宣伝ではなく、能力と限界を同じ重みで示す文書です。

取り違いやすい点のまとめ [1.2.b]

この分野の選択肢は、正しい文の一部だけを入れ替えて作られ、次の表は取り違いの典型と実際の理解を示す。

よくある取り違え	実際の理解
公開コードの Block で機密ファイルを守れる	Block は提案が公開コードと一致するのを抑える機能。機密の文脈を外すのはコンテンツ除外
コンテンツ除外はすべてのプランで使える	Copilot Business と Enterprise の機能
コンテンツ除外で漏れが必ずゼロになる	間接的な意味情報、シンボリックリンク、Chat の Edit・Agent モードなどに限界がある
Copilot code review の承認で必須の承認を満たせる	既定は Comment のレビューで、必須の承認に数えられない。承認の設定は既定で無効
Block にすればクラウドエージェントの提案に一致は出ない	クラウドエージェントと CLI は、Block でも一致を生成することがある
テストが通れば生成コードは安全	テストは網羅しているとは限らない。code scanning とレビューが別に要る
幻覚は PR の文章補完では起きない	言語モデルが作る文章なので、幻覚は起きる
<code>--allow-all-tools</code> を付ければ安全に自動化できる	同じ権限で、事前の承認なしにコマンドが実行される
クラウドエージェントは既定のブランチへ直接コミットできる	書き込めるのは <code>copilot/</code> ブランチか、呼ばれた PR のブランチだけ

サンプルはここまでです

続き(第2章～巻末)は、GitHub Copilot GH-300 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

GitHub Copilot GH-300 学習用テキスト(無料サンプル)

版

2026年10月版(Microsoft Learn「GH-300: GitHub Copilot 学習ガイド」Skills measured as of August 7, 2026(2026-10-13 取得)準拠)

発行

IT資格道場(<https://www.shikaku-doujou.com>)

お問い合わせ

info@shikaku-doujou.com

GitHub、GitHub Copilot は GitHub, Inc. の商標または登録商標です。Microsoft および Visual Studio は Microsoft Corporation の商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式の出題範囲(GH-300 学習ガイド・2026年8月7日現在のスキル)と GitHub Docs などの公開文書を基に作成したもので、GitHub, Inc.・Microsoft との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。