

付録2：実験データと評価結果（一部）

- ・NoはデータのユニークIDです。
- ・文脈的に合致した同じ文章をtext1とtext2の両方に挿入し、文章の長文化をしたもののセットを長文化セット列で表し値が一致するものがセットです。長文化種別は、元の短文化か、長文化後の文章かを表しています。
- ・text1,text2は類似度を測る対象の文章です。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1、別=0で表しています。
- ・text-embedding-3-smallから、ChatGPTスコアは各モデルでの類似度評価結果を表します。
- ・text-embedding系モデルの2種は0~1で表し、1に近いほど類似度が高いです。
- ・ChatGPT系モデルの2種は0~100で表し、100に近いほど類似度が高いです。
- ・ChatGPT系モデルは評価値の根拠も同時に出力したため、掲載しています。

No	長文化種別	長文化セット	text1	text2	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	ChatGPT比較類似度の根拠	ChatGPT独立類似度の根拠
1	—	—	アジャイル開発は反復的で柔軟性が高く、変更に対応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイル開発は反復的で柔軟性があり、変化に迅速に対応できるのに対し、ウォーターフォール開発は計画に基づいた直線的なプロセスで、各工程を順に完了させてから次に進む点が大きな違いです。	1	0.97910	0.98436	95	95	両方の文はアジャイル開発とウォーターフォール開発の違いを説明しており、意味合いはほぼ3です。どちらもアジャイル開発の柔軟性とウォーターフォール開発の計画重視を強調しています。表現の違いはあるものの、伝えたい内容は一致しています。	両方の文は、アジャイル開発とウォーターフォール開発の特徴の違いを説明しています。具体的には、アジャイル開発の反復性と柔軟性、変化への対応力を強調し、ウォーターフォール開発の計画重視と直線的なプロセスを指摘しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。
2	—	—	アジャイル開発は反復的で柔軟性が高く、変更に対応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイルは反復的で柔軟性が高く、変更対応を重視します。一方、ウォーターフォールは各工程を順序立てて進め、後戻りが難しい計画重視の手法です。	1	0.91004	0.89423	95	85	両方の文はアジャイル開発とウォーターフォール開発の特徴を説明し、それぞれの開発手法の違いを強調しています。アジャイルの「反復的」「柔軟性」「変更対応」とウォーターフォールの「計画重視」「順序立て」「後戻りが難しい」という点が共通しており、意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の特徴を比較しており、アジャイルの反復性と柔軟性、ウォーターフォールの計画重視と直線的な進行について言及しています。違いを強調する点で内容が非常に類似していますが、表現の仕方や細部に若干の違いがあります。
3	—	—	アジャイル開発は反復的で柔軟性が高く、変更に対応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	ウォーターフォールモデルは工程が順序立てられているため、柔軟性に欠け、要件変更や市場変化への対応が困難でした。この課題を解決するため、**反復的かつ適応的な開発**を重視するアジャイル開発が誕生しました。アジャイルは顧客との連携や短期間の反復（スプリント）を通じて、変化に対応しながら継続的に価値を提供する手法として進化しました。	0	0.76930	0.73989	85	85	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てており、特に柔軟性や適応性の違いを強調しています。文2はアジャイル開発の誕生背景を詳しく述べているため、文1よりも詳細ですが、基本的な意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てています。特に、アジャイル開発の柔軟性と反復性、ウォーターフォール開発の計画重視と柔軟性の欠如について述べています。両者ともアジャイル開発がウォーターフォール開発の課題を解決するために生まれたことを示唆しており、基本的な意味合いは非常に類似しています。ただし、文2はアジャイル開発の進化や顧客との連携についても詳しく述べているため、若干の情報の違いがあります。
4	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`/.ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：ユーザーは非対称暗号方式を用いて「公開鍵」と「秘密鍵」のペアを生成します。秘密鍵は安全に保管し、公開鍵は認証サーバや接続先に共有します。2. **公開鍵の登録**：ユーザーは公開鍵を接続先サーバに登録します。登録された公開鍵は、サーバ上の認証ファイル（例：`/.ssh/authorized_keys`）に保存されます。3. **認証の開始**：ユーザーがサーバに接続を試みると、サーバは認証の一環としてランダムなデータ（チャレンジ）をユーザーに送信します。4. **秘密鍵による署名**：ユーザーは受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵を用いて署名を検証します。このプロセスで、ユーザーが秘密鍵の正当な所有者であることが確認されます。6. **接続の許可**：認証が成功すると、サーバは接続を許可し、安全な通信が確立されます。この手順により、パスワードを問わずに高いセキュリティで認証が行われます。	1	0.97842	0.96964	98	95	両方の文章は、公開鍵認証の手順を説明しており、伝えたい内容や意味はほぼ同一です。細かい表現の違いはあるものの、全体としての意味合いに大きな差異はありません。	両方の文は公開鍵認証の手順について詳細に説明しており、手順の流れや内容が非常に類似しています。鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可といったステップが順序で記載されています。違いは、用語の選択や表現のわずかな違いに過ぎず、全体の意味合いにはほとんど影響を与えません。したがって、意味合い的な類似度は非常に高いと評価できます。
5	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`/.ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. ユーザーが鍵ペア（公開鍵と秘密鍵）を生成します。2. 公開鍵をサーバに登録します。3. ユーザーが接続を試みると、サーバは暗号化されたチャレンジを送信します。4. ユーザーは秘密鍵でチャレンジを復号し、応答を返します。5. サーバが応答を確認し、認証を完了します。	0	0.89827	0.87419	90	85	両方の文章は公開鍵認証の基本的な手順を説明しており、意味合いとしては非常に類似しています。どちらも鍵ペアの生成、公開鍵の登録、チャレンジと応答のプロセスを通じて認証が行われることを述べています。違いは、入力文1がより詳細で技術的な説明を含んでいる点です。	両方の文は公開鍵認証の手順について説明しており、基本的な流れは非常に似ています。鍵ペアの生成、公開鍵の登録、チャレンジの送信と応答、そして認証の完了という主要なステップが共通しています。ただし、入力文1は手順をより詳細に説明しており、特に「署名」と「署名の検証」という用語を使用しているのに対し、入力文2は「復号」と「応答を返す」という表現を用いています。この違いが若干の意味合いの違いを生んでいますが、全体としては3プロセスを指しているため、高い類似度があります。
6	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`/.ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. サーバがユーザーに暗号化されたチャレンジを送信します。2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバに渡そうとします（実際は公開鍵を登録します）。3. サーバは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。4. ユーザーが秘密鍵を使用して復号し、登録プロセスをやり直します。5. 最終的に公開鍵がサーバに登録され、接続が試みられます。	0	0.88345	0.83765	60	45	両方の文章は公開鍵認証のプロセスを説明していますが、具体的な手順や用語の使い方に違いがあります。入力文1は正確な手順を説明しているのに対し、入力文2は一部の手順が誤解を招く表現になっており、特に秘密鍵の取り扱いに関する記述が異なります。これにより、全体的な意味合いの違いが生じています。	両方の文は公開鍵認証の手順について説明していますが、手順の詳細や順序の違いがあります。入力文1は、鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可という具体的なステップを順序立てて説明しています。一方、入力文2は手順が曖昧で、特に秘密鍵の取り扱いに誤解を招く表現があります（秘密鍵をサーバに渡そうとするなど）。また、手順の順序も異なり、鍵ペアの生成が認証プロセスの途中で促されるなど、実際の公開鍵認証の流れとは異なっています。したがって、意味合い的な類似度は中程度と評価しました。

7	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なるアプローチを採用しており、柔軟なスキーマ設計と高いスケラビリティを特徴とする非リレーショナルデータベースです。	1	0.90452	0.89878	95	95	両方の文はNoSQLの特徴を説明しており、非リレーショナルデータベースであること、柔軟なスキーマ設計、スケラビリティの高さを強調しています。意味合いはほぼ3ですが、入力文2はリレーショナルデータベースとの対比を明示している点で若干の違いがあります。	両方の文は、NoSQLが非リレーショナルデータベースであること、柔軟なスキーマ設計を持つこと、高いスケラビリティを特徴とすることを述べています。文2は、リレーショナルデータベースとの違いを明示していますが、全体的な意味合いは非常に類似しています。
8	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なり、柔軟なデータモデルを採用したデータベースの総称です。スケラビリティや高速度性に優れ、文書型、キー・値型、カラム型、グラフ型などの種類があります。	0	0.83348	0.82690	85	85	両方の文はNoSQLの特徴として「柔軟性」と「スケラビリティ」を挙げており、NoSQLがリレーショナルデータベースとは異なることを強調しています。入力文2はさらに「高速度性」や具体的なデータモデルの種類についても触れていますが、全体的な意味合いは非常に類似しています。	両方の文はNoSQLデータベースについて述べており、非リレーショナルデータベースであること、柔軟なデータモデルを採用していること、スケラビリティに優れていることを共通して説明しています。入力文2はさらにNoSQLの種類や高速度性についても言及していますが、基本的な特徴に関する情報は一致しているため、高い類似度があります。
9	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	SQLに「No」を突き付ける人々は、スケラビリティや柔軟性を重視し、NoSQL（非リレーショナルデータベース）を支持する傾向があります。彼らは、大量データ処理や動的スキーマ、水平分散が必要なユースケース（例：リアルタイム分析や分散型アプリ）でSQLの制約を問題視します。MongoDBやCassandraなどのNoSQLは、特定の用途でSQLを補充・代替する選択肢として注目されています。	0	0.71857	0.74304	75	85	両方の文はNoSQLの特徴である「スケラビリティ」や「柔軟性」に言及しており、NoSQLがリレーショナルデータベースであることを共通して述べています。ただし、入力文2はNoSQLを支持する理由や具体的なユースケースについても詳しく述べているため、意味合い的にはより広範な内容を含んでいます。基本的なテーマは一致しています。	両方の文はNoSQLデータベースに関する内容であり、特にスケラビリティや柔軟なスキーマ設計の利点を強調しています。入力文1はNoSQLの特徴を簡潔に述べており、入力文2はNoSQLを支持する理由や具体的なユースケースを詳しく説明しています。内容の焦点が一致しており、NoSQLの利点に関する理解が共通しているため、高い類似度があります。ただし、入力文2はSQLとの比較や具体例を含み、詳細度に差があります。
10	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	要件定義や設計段階での問題が原因である可能性が高いです。十分に練られていない要件定義や設計上のミスが、テスト工程における障害の多発に直結します。	1	0.92471	0.91897	95	95	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であるという意味を伝えています。表現の違いはあるものの、伝えたい内容はほぼ同一です。	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であることを述べています。言い回しや表現に若干の違いはありますが、伝えたい内容や論点は非常に類似しています。
11	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	ウォーターフォール開発でシステムテスト工程で障害が多発する場合、主に**要件定義**や**設計**に問題がある可能性が高いです。要件の不備や曖昧さが原因で正しい仕様が実装されない場合や、設計段階でのミスがテスト工程で顕在化します。また、**結合テスト**での検証不足や、単体テストでの抜け漏れも要因となります。	0	0.75388	0.80472	85	85	両方の文は、要件定義や設計の不備がテスト工程での障害多発の原因であることを示しています。入力文2はより具体的な開発プロセス（ウォーターフォール開発）やテスト工程の詳細を含んでいますが、基本的な意味合いは一致しています。	両方の文は、要件定義や設計工程に問題があるとテストで障害が多発する可能性が高いことを述べています。入力文1は要件定義や設計ミスが直接テストでの障害につながることを述べており、入力文2はウォーターフォール開発におけるシステムテストでの障害多発の原因として要件定義や設計の問題を挙げています。入力文2はさらに結合テストや単体テストの不足も要因として挙げており、詳細な説明が加えられていますが、基本的な意味合いは非常に類似しています。
12	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	システムで障害が多発するケースには、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障が挙げられます。特にスケラビリティを考慮しない設計や負荷テスト不足は、アクセス急増時に問題を引き起こします。また、運用プロセスの不備や変更管理の失敗、冗長性の欠如も影響します。継続的な監視や定期的なメンテナンスが重要です。	0	0.65997	0.53354	60	60	両方の文はシステムやプロジェクトにおける問題の原因について述べていますが、入力文1は要件定義や設計ミスに特化しており、入力文2はより広範な原因をカバーしています。共通点として設計の不備が挙げられますが、入力文2は他の要因も含むため、意味合い的な類似度は中程度です。	両方の文はシステムにおける障害の原因について述べていますが、焦点が異なります。入力文1は主に要件定義や設計工程の問題に焦点を当てており、これがテストでの障害多発につながるとしています。一方、入力文2はシステム障害の原因をより広範囲に捉え、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障などを挙げています。特にスケラビリティや運用プロセスの不備についても言及しています。共通点として設計の問題が挙げられますが、入力文2はより多くの要因を考慮しており、具体的な対策についても触れています。したがって、類似度は中程度と評価しました。
13	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社で物理サーバーを所有し運用する形式であり、一方クラウドはインターネットを通じて必要なリソースを利用する形態です。	1	0.95975	0.95273	95	95	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、意味合いは非常に近いです。どちらもオンプレミスが自社でのサーバー管理を指し、クラウドがインターネット経由でのリソース利用を指しているため、伝えたい内容はほぼ同一です。表現の違いはあるものの、意味合いはほぼ一致していません。	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、内容的に非常に類似しています。どちらもオンプレミスが自社で物理サーバーを所有・管理する形式であることを、クラウドがインターネット経由でリソースを利用する形式であることを述べています。言い回しに若干の違いはありますが、意味合いはほぼ一致しています。
14	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社でサーバーやインフラを構築・管理する方式で、初期投資や保守が必要です。一方、クラウドはインターネット経由で外部サービスを利用する方式で、スケラビリティやコスト効率に優れます。	0	0.82750	0.76366	85	85	両方の文はオンプレミスとクラウドの違いを説明しており、基本的な意味合いは非常に似ています。オンプレミスは自社での管理が必要であること、クラウドはインターネット経由で利用することを述べています。入力文2は追加でスケラビリティやコスト効率についても言及していますが、全体的な意味合いは一致しています。	両方の文は、オンプレミスとクラウドの違いを説明しています。オンプレミスについては、自社でサーバーを保有・管理する点が共通しており、クラウドについてはインターネット経由でリソースを利用する点が一致しています。違いとしては、入力文2が初期投資やスケラビリティ、コスト効率についても言及しているため、詳細度に差がありますが、基本的な意味合いは非常に類似しています。
15	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	クラウド移行は、スケラビリティ、コスト効率、災害対策の強化、運用負荷の軽減を実現します。リソースを柔軟に拡張でき、初期投資を抑えつつ、セキュリティや最新技術の恩恵を享受可能です。競争力向上に寄与します。	0	0.44651	0.34687	40	30	両方の文はクラウドに関連していますが、入力文1はオンプレミスとクラウドの基本的な違いを説明しているのに対し、入力文2はクラウド移行の具体的な利点に焦点を当てています。意味合いには関連性がありますが、焦点が異なるため、意味合い的な類似度は中程度です。	入力文1はオンプレミスとクラウドの基本的な違いを説明しており、主に技術的な形態に焦点を当てています。一方、入力文2はクラウド移行の利点について述べており、スケラビリティやコスト効率、競争力向上などの具体的なメリットに焦点を当てています。両者はクラウドに関連する内容を含んでいるものの、焦点が異なるため、意味合い的な類似度は低いです。
16	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデータを体系的に管理するシステム全体を指し、複数のテーブルを含む容器のようなものです。一方、テーブルはデータベース内でデータを行と列で構造化して格納する単位です。	1	0.89072	0.87697	95	95	両方の文はデータベースとテーブルの関係を説明しており、意味合いは非常に似ています。どちらもデータベースをデータを管理するシステム全体として説明し、テーブルをデータを構造化して格納する単位として説明しています。入力文2の方が若干詳細ですが、基本的な意味合いは一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であることを、テーブルがデータを構造化して格納する単位であることを述べています。両者は「複数のテーブルを管理し」と表現しており、文1よりも少し詳細ですが、基本的な意味合いは非常に類似しています。

17	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースは、データを体系的に管理・保存するための**全体の容器**や構造を指します。一方、テーブルはデータベース内でデータを格納するための**個々の構造化された単位**です。データベースは複数のテーブルを持つことができ、それぞれのテーブルが行と列を持ち、特定の種類のデータを持っています。データベースはシステム全体、テーブルはその中の部品と考えられます。	0	0.86902	0.82861	85	95	両方の文はデータベースとテーブルの役割と関係性を説明しており、基本的な意味合いは一致しています。入力文2はより詳細で具体的な説明を提供していますが、全体的な意味は入力文1と一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であること、テーブルがその中でデータを構造化して保存する単位であることを述べています。入力文2は、入力文1の内容をより詳細に説明しており、データベースが複数のテーブルを持つことや、テーブルが行と列を持つことについても触れています。全体として、両方の文は概念を伝えており、意味合い的に非常に類似しています。
18	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデジタル情報を整理・管理するシステムで、データを効率的に保存・検索・操作できます。一方、テーブルは物理的な家具で、物を置く・作業する場を提供します。データベースは抽象的で機能的、テーブルは具体的な物理的な用途に特化しています。	0	0.79866	0.67894	40	40	両方の文は「データベース」と「テーブル」に関する情報を提供していますが、文1はデータベース内のテーブルの役割を説明しているのに対し、文2はデータベースとテーブルを異なる文脈で比較しています。文2では、データベースは情報管理システムとしての機能を強調し、テーブルは物理的な家具としての役割を説明しており、意味合い的には異なる内容を持っています。	両方の文は「データベース」と「テーブル」という用語を扱っていますが、それぞれの文での「テーブル」の意味が異なります。入力文1では、データベース内の構造化されたデータの保存単位としての「テーブル」を説明していますが、入力文2では物理的な家具としての「テーブル」を説明しています。データベースに関する説明は類似していますが、テーブルに関する説明が異なるため、全体としての意味合い的類似度は低くなります。
19	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合するテーブルの両方で一致する行のみを取得します。一方、**OUTER JOIN**は一致する行に加えて、一方のテーブルにしか存在しない行も含めることができます。	1	0.85767	0.87835	95	95	両方の文はINNER JOINとOUTER JOINの機能の違いを説明しており、基本的な意味合いは非常に近いです。INNER JOINが両テーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルに一致しないデータも含めるといった点で、意味の一致が見られます。表現の違いはあるものの、伝えたい内容はほぼ3です。	両方の文は、データベースにおけるINNER JOINとOUTER JOINの機能を説明しています。内容としては、INNER JOINが両方のテーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルに一致しないデータも含めると述べています。言い回しや細かい表現の違いはありますが、伝えてい情報はほぼ3であるため、高い類似度スコアを付きました。
20	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合条件を満たす両方のテーブルの共通部分のみを取得します。一致しないデータは結果に含まれません。 **OUTER JOIN**は、条件を満たさないデータも含める結合で、以下の種類があります： - **LEFT OUTER JOIN**：左側のテーブルの全データと一致する右側のデータ。 - **RIGHT OUTER JOIN**：右側のテーブルの全データと一致する左側のデータ。 - **FULL OUTER JOIN**：両方のテーブルの全データを取得します。	0	0.81617	0.80728	85	85	両方の文はINNER JOINとOUTER JOINの基本的な概念を説明しており、意味合いは非常に似ています。ただし、入力文2はOUTER JOINの種類についての追加情報を提供しているため、完全に同一の意味合いではありません。	両方の文は、INNER JOINとOUTER JOINの基本的な概念を説明しており、特にINNER JOINについては非常に類似した内容を述べています。OUTER JOINに関しては、入力文2がLEFT、RIGHT、FULL OUTER JOINの詳細を追加しているため、情報量が多く、やや異なる視点を提供していますが、基本的な意味合いは一致しています。したがって、全体的な意味合い的類似度は高いと評価できます。
21	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	INNER JOINをOUTER JOINに書き換えるには、結合タイプを明示的に指定する必要があります。INNER JOINでは共通データのみ取得しますが、OUTER JOINでは片方のテーブルの全行を保持します。	0	0.82297	0.72906	80	85	両方の文はINNER JOINとOUTER JOINの違いについて説明していますが、入力文2は書き換え方法についても触れており、意味合いに若干の追加情報があります。それでも、基本的な意味合いは非常に似ています。	両方の文はINNER JOINとOUTER JOINの違いについて説明しています。入力文1は、INNER JOINが両テーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルの全行を保持することを説明しています。ただし、入力文2はINNER JOINをOUTER JOINに書き換える方法についても触れており、結合タイプの指定についての情報が追加されています。このため、内容の焦点が若干異なる部分があるものの、基本的な意味合いは非常に類似しています。
22	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	用途によりですが、クラウドは初期コストが抑えられ柔軟性が高い一方で、長期的には利用状況に応じてコストが増加する可能性があります。	1	0.88793	0.93630	95	95	両方の文はクラウドの利点と欠点を述べており、初期コストの低さと柔軟性の高さを利点として挙げ、長期的なコスト増加の可能性を欠点として指摘しています。意味合いはほぼ3ですが、文2では「可能性があります」と若干の不確実性を示しています。	両方の文は、クラウドの特性について述べており、初期コストの低さと柔軟性の高さ、そして長期的なコスト増加の可能性について言及しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。
23	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	長期的なコスト効率は利用状況によりですが、一般的に次のように考えられます： - **クラウド**：初期投資が少なく、使用量に応じた課金モデルのため、柔軟性が高くスケラブルな運用が可能。ただし、長期で大規模な利用ではランニングコストが増える場合も。 - **オンプレミス**：初期投資が大きい一方、長期間にわたって一定規模の利用を続ける場合、運用コストが安定し、総コストが抑えられることがあります。ただし、保守・更新コストが追加される点に留意。必要な規模や用途によってどちらが効率的かが異なります。	0	0.78180	0.72671	75	85	両方の文はクラウドの初期コストの低さと長期的なコスト増加の可能性について述べています。意味合いとしては、クラウドのコストに関する利点と欠点を説明している点で類似していますが、入力文2はオンプレミスの点と比較も含んでおり、より広範な視点を提供しています。	両方の文はクラウドのコストと柔軟性に関する内容を扱っており、特に初期コストの低さと利用量に応じたコスト増加について言及しています。入力文2はオンプレミスについても触れていますが、クラウドに関する部分は入力文1と非常に類似しています。したがって、意味合い的な類似度は高いと評価できます。
24	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	オンプレミスは、一度の設備投資で長期間安定稼働が可能で、クラウドのような毎月のランニングコストが発生しません。大容量データや高負荷のアプリケーションでも、自社内で管理することで追加コストを抑えられます。特に既存設備の活用やカスタマイズ性が求められる環境では、オンプレミスが最も合理的で経済的な選択です！	0	0.86646	0.58247	50	65	両方の文章はITインフラの選択肢に関する情報を提供していますが、焦点が異なります。入力文1はクラウドの特性に焦点を当てており、入力文2はオンプレミスの利点を強調しています。どちらもコストと柔軟性に関する情報を含んでいますが、比較の対象が異なるため、意味合い的には部分的にしか重なりません。	両方の文はITインフラの選択肢であるクラウドとオンプレミスの特徴を説明しています。入力文1はクラウドの利点とコストの増加について述べており、入力文2はオンプレミスの利点とコスト管理について述べています。どちらもコストと柔軟性に関する情報を含んでおり、ITインフラの選択に関する議論の一部を形成していますが、具体的な内容や視点が異なるため、完全に一致するわけではありません。
25	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確に定義します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。各工程を完了後に次へ進むため、工程の見直しは手間がかかります。	ウォーターフォール開発は、工程を順番に進める直線的な開発モデルです。主な流れは以下の通りです。 1. **要件定義**：システムの要件や仕様を明確に定義します。 2. **設計**：システム全体の構造を設計し、詳細な設計も行います。 3. **実装**：設計内容に基づいてプログラムを開発します。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用と保守**：システムをリリースし、運用や問題修正を行います。 各工程が完了してから次に進む仕組みのため、途中で工程を見直す際には大きな手間がかかります。	1	0.98439	0.97103	98	95	両方の文章はウォーターフォール開発の特徴と工程を説明しており、意味合いに大きな違いは有りません。「要求定義」や「要件定義」などの違い（例：「要求定義」vs「要件定義」）や表現の微細な違いはありますが、伝えたい内容はほぼ3です。	両方の文はウォーターフォール開発モデルについて説明しており、工程の順序や内容がほぼ3です。用語の違いや「要求定義」「要件定義」などや表現のわずかな違いはありますが、全体的な意味合いは非常に類似しています。工程の進め方や見直しの手間についても同様の説明がされています。

26			ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進行します：1. **要件定義**：システムの要件や仕様を明確化する。 2. **基本設計**：システム全体の構造や外部仕様を設計する。 3. **詳細設計**：プログラム単位の具体的な設計を行う。 4. **実装（コーディング）**：設計に基づきプログラムを作成する。 5. **単体テスト**：個々のプログラムやモジュールをテストする。 6. **結合テスト**：モジュール同士を結合して動作を確認する。 7. **システムテスト**：システム全体の動作を確認し、要件通りか検証する。 8. **運用・保守**：完成したシステムを運用し、問題が発生した場合に修正する。各工程が順に完了しないと次に進めないのが特徴です。	0	0.93794	0.93866	90	85	両方の文章はウォーターフォール開発の基本的な考え方と工程の流れを説明しており、意味合いは非常に似ています。どちらも工程を順序立てて進めることの重要性を強調しているため、入力文2は工程をより詳細に説明しているため、若干の違いがあります。	両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要求定義から運用・保守までの流れを説明しており、全体的なプロセスの流れは非常に似ています。ただし、入力文2はより詳細な工程（基本設計、詳細設計、単体テスト、結合テスト、システムテスト）を含んでおり、工程の細分化がされています。この違いがスコアに影響していますが、全体的な意味合いは非常に類似しています。
27			ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進行します：5. **単体テスト**：個々のプログラムやモジュールをテストする。 1. **要件定義**：システムの要件や仕様を明確化する。 7. **システムテスト**：システム全体の動作を確認し、要件通りか検証する。 3. **詳細設計**：プログラム単位の具体的な設計を行う。 8. **運用・保守**：完成したシステムを運用し、問題が発生した場合に修正する。 4. **実装（コーディング）**：設計に基づきプログラムを作成する。 6. **結合テスト**：モジュール同士を結合して動作を確認する。 2. **基本設計**：システム全体の構造や外部仕様を設計する。 各工程が順に完了しないと次に進めないのが特徴です。	0	0.91345	0.91544	90	85	両方の文はウォーターフォール開発モデルの基本的な概念とその工程について説明しています。両方とも、工程が順序立てて進行し、各工程が完了しないと次に進めないという特徴を強調しています。意味合い的には非常に類似していますが、入力文2はテスト工程をより詳細に説明しており、若干の違いがあります。それでも、全体的な意味合いはほぼ3です。	両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要件定義、設計、実装、テスト、運用・保守といった主要な工程を含んでおり、全体的な流れも似ています。ただし、入力文2はテスト工程より詳細に分けて説明しており、工程の順序が異なる部分もありますが、このため、完全に一致するわけではありませんが、基本的な概念と流れは非常に類似しています。
28			Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、.javaファイルとして保存。2. **コンパイル**：javacコンパイラを使用してソースコードをコンパイルし、.classファイルに変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムの実行手順は以下の通りです：1. **ソースコード作成**：Javaでプログラムを記述し、.java 拡張子のファイルとして保存します。2. **コンパイル**：javacコンパイラを使ってソースコードをバイトコード(.classファイル)に変換します。3. **クラスローダーによるロード**：JVM (Java仮想マシン) がバイトコードを読み込み、メモリ上に展開します。4. **バイトコードの実行**：JVM内でインタプリタまたはJIT (Just-In-Time) コンパイラがバイトコードをネイティブコードに変換し、実行します。5. **結果表示**：プログラムの出力が表示されます。このプロセスにより、Javaはプラットフォームに依存せず動作する特性を持っています。	1	0.90186	0.95154	98	95	両方の文は、Javaプログラムの実行手順を説明しており、内容的にはほぼ同一です。細かい表現の違い（例：「動作順序 vs 実行手順」、「結果表示 vs 結果の表示」）はありますが、全体の意味合いに大きな違いはありません。	両方の文は、Javaプログラムの実行手順について詳細に説明しており、内容的にはほぼ同一です。手順の順序や内容も一致しており、用語の違い（例えば「動作順序」と「実行手順」）や表現のわずかな違い（例えば「結果表示」と「結果の表示」）があるものの、意味合いに大きな違いはありません。したがって、意味合い的な類似度は非常に高いと評価できます。
29			Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、.javaファイルとして保存。2. **コンパイル**：javacコンパイラを使用してソースコードをコンパイルし、.classファイルに変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムが動作する基本的な順序は以下の通りです：1. **ソースコード作成**：Javaファイル(.java)にソースコードを記述します。2. **コンパイル**：javacコンパイラを使用して、ソースコードをバイトコード(.classファイル)に変換します。例：javac MyProgram.java 3. **クラスローダーによるロード**：Java Virtual Machine (JVM)が必要な.classファイルを読み込みます。4. **バイトコードの実行**：JVMのバイトコードの安全性を確認します。5. **実行準備**：JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。6. **実行（メインメソッドの呼び出し）**：実行のエントリーポイントであるmainメソッドが呼び出され、プログラムが開始します。例：java MyProgram 7. **コード実行**：JVMがバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。8. **終了**：プログラムが終了するまでメモリを解放し、プロセスを終了します。この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。	0	0.92306	0.92074	90	85	両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。両方ともJavaのプラットフォーム非依存性を強調し、ソースコードの作成から実行までの流れを説明しています。入力文2はより詳細な説明を含んでいますが、全体的な意味合いは一致しています。	両方の入力文は、Javaプログラムの動作順序について説明しており、基本的な意味合いは非常に似ています。具体的には、ソースコードの作成、コンパイル、クラスローダーによるロード、バイトコードの実行といった主要なステップが共通しています。ただし、入力文2にはバイトコードの検証や実行準備、プログラム終了時のメモリ解放といった詳細なステップが追加されており、より詳細なプロセスが記述されています。このため、全体的な流れは似ているものの、細部において差異があるため、完全に同一の意味合いとは言えません。
30			Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、.javaファイルとして保存。2. **コンパイル**：javacコンパイラを使用してソースコードをコンパイルし、.classファイルに変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムが動作する基本的な順序は以下の通りです：1. **実行準備**：JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。2. **ソースコード作成**：Javaファイル(.java)にソースコードを記述します。3. **終了**：プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。4. **クラスローダーによるロード**：Java Virtual Machine (JVM)が必要な.classファイルを読み込みます。5. **コンパイル**：javacコンパイラを使用して、ソースコードをバイトコード(.classファイル)に変換します。例：javac MyProgram.java 6. **バイトコードの検証**：JVMのバイトコードバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。7. **コード実行**：JVMがバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。8. **実行（メインメソッドの呼び出し）**：実行のエントリーポイントであるmainメソッドが呼び出され、プログラムが開始します。例：java MyProgram この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。	0	0.89679	0.87155	80	75	両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。どちらもJavaがプラットフォームに依存せず動作することを強調しています。ただし、入力文2はより詳細な手順を含んでおり、特定のプロセス（例：バイトコードの検証、メインメソッドの呼び出し）についての追加情報を提供しています。これにより、意味合い的な細部に関する違いがありますが、全体的な意味合いは一致しています。	両方の文はJavaプログラムの動作順序について説明していますが、詳細なステップや順序の違いがあります。入力文1は、ソースコード作成から結果表示までの基本的な流れを簡潔に説明しています。一方、入力文2は、実行準備やバイトコードの検証、メインメソッドの呼び出しなど、より詳細なステップを含んでいます。両方の文がJavaのプラットフォーム非依存性に言及している点は共通していますが、具体的な手順の記述に違いがあるため、完全に一致するわけではありません。したがって、意味合い的な類似度は75と評価しました。
31			ウォーターフォール開発は以下の順序で進行します：1. **要件定義**：システムの要件を明確化。 2. **基本設計**：全体の構造を設計。 3. **詳細設計**：プログラム単位の具体設計。 4. **実装**：設計に基づきプログラム作成。 5. **単体テスト**：各プログラムをテスト。 6. **結合テスト**：モジュール間の動作確認。 7. **システムテスト**：システム全体を検証。 8. **運用・保守**：問題修正や改善を行う。順序を厳密に守ることが特徴です。	ウォーターフォール開発は以下の順序で進行します：3. **詳細設計**：プログラム単位の具体設計。 7. **システムテスト**：システム全体を検証。 1. **要件定義**：システムの要件を明確化。 6. **結合テスト**：モジュール間の動作確認。 4. **実装**：設計に基づきプログラム作成。 8. **運用・保守**：問題修正や改善を行う。 2. **基本設計**：全体の構造を設計。 5. **単体テスト**：各プログラムをテスト。 順序を厳密に守ることが特徴です。	0	0.96852	0.95236	95	85	両方の文はウォーターフォール開発のプロセスとその特徴である「順序を厳密に守ること」を説明しています。ステップの順序が異なるものの、伝えたい内容や意味はほぼ3です。	両方の文はウォーターフォール開発のプロセスを説明しており、3ステップを含んでいます。しかし、入力文2ではステップの順序が異なっており、ウォーターフォール開発の特徴である「順序を厳密に守ること」が反映されていません。このため、意味合い的には類似していますが、順序の重要性が異なるため完全には一致しません。

32	—	<p>Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成** **Javaでコードを記述し、.javaファイルとして保存。2. **コンパイル** **javacコンパイラを使用してソースコードをコンパイルし、バイトコード(.classファイル)に変換。3. **クラスローダーによる読み込み** **JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行** **JVMでインタープリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示** プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。</p>	<p>Javaプログラムが動作する基本的な順序は以下の通りです：1. **実行準備** JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。2. **ソースコード作成** Javaファイル(.java)にソースコードを記述します。3. **終了** プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。4. **クラスローダーによるロード** Java Virtual Machine (JVM)が必要な.classファイルをメモリにロードします。5. **コンパイル** Javacコンパイラを使用して、ソースコードをバイトコード(.classファイル)に変換します。例：javac MyProgram.java。6. **バイトコードの検証** JVMのバイトコードベリファイアがコードの安全性を確認します。7. **コード実行** JVMがバイトコードを解釈（またはJIT keys）し、各命令をCPUで実行します。8. **実行（メインメソッドの呼び出し）** 実行のエントリーポイントであるmainメソッドが呼び出され、プログラムが開始します。例：java MyProgram この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。</p>	0	0.89726	0.87175	80	75	<p>両方の文章はJavaプログラムの実行プロセスを説明しており、基本的な意味合いは3です。どちらもJavaのプラットフォーム非依存性を強調しています。ただし、入力文2はより詳細な手順を含んでおり、特に「実行準備」や「バイトコードの検証」など、入力文1にはない具体的なプロセスが追加されています。これにより、意味合いに若干の違いが生じていますが、全体としては3テーマを扱っています。</p>
33	—	<p>公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成** 利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵をサーバーや他者に共有します。2. **公開鍵の登録** 利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：/ssh/authorized_keys）に保存されます。3. **認証プロセスの開始** 利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名** 利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証** サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可** 認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。</p>	<p>公開鍵認証の手順は以下の通りです：1. サーバーがユーザーに暗号化されたチャレンジを送信します。2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバーに渡そうとします（実際は公開鍵を登録します）。3. サーバーは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。4. ユーザーが秘密鍵を使用して暗号化し、登録プロセスをやり直します。5. 最終的に公開鍵がサーバーに登録され、接続が試みられます。</p>	0	0.88341	0.83840	50	45	<p>両方の文章は公開鍵認証のプロセスを説明しようとしていますが、入力文2には誤解を招く表現や不正確な情報が含まれています（例：秘密鍵をサーバーに渡そうとするなど）。そのため、意味合い的には一致していない部分が多く、類似度は低くなります。</p>
34	—	<p>### 要件定義の手順1 ### プロジェクトの背景と目的の明確化 プロジェクトのコンセプトやビジョンを明確に定義する。2. ###利害関係者の特定 プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. ###要件の収集 利害関係者から要件を収集し、優先順位を付ける。4. ###要件の分析 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. ###優先順位の付与 要件に優先順位を与え、重要なものとそうでないものを区別する。6. ###要件仕様書の作成 収集された要件をまとめて、要件仕様書として文書化する。7. ###要件の確認と承認 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。</p>	<p>### 要件定義の流れ1 ### プロジェクトの背景理解 プロジェクトの目的、スコープ、ステークホルダーを理解する。2. ###要件収集 ステークホルダーとのコミュニケーションを通じて、必要な機能や要件を収集する。3. ###要件分析 収集した要件を整理し、矛盾点や抜け漏れを特定する。4. ###要件の整理と整合性の確認 複数の要件を整理し、矛盾や重複を解消して要求事項を洗練させる。5. ###要件文書の作成 要求事項を記述した要件仕様書を作成し、関係者と共有する。6. ###要件の検証 作成された要件がプロジェクトの目的やスコープと整合しているかを確認する。7. ###フィードバックと調整 ステークホルダーからのフィードバックを受け、必要に応じて要件を調整する。8. ###最終的な要件の確認 最終的な要件がすべてのステークホルダーによって承認される。</p>	1	0.81901	0.91259	90	85	<p>両方の文章は、要件定義のプロセスを説明しており、プロジェクトの背景理解、要件の収集と分析、文書化、確認と承認といった主要な目的や意味合いが非常に似ています。細部の表現やステークホルダーとの関係の強調に若干の違いがありますが、全体的な意味合いはほぼ一致しています。</p>
35	—	<p>### 要件定義の手順1 ### プロジェクトの背景と目的の明確化 プロジェクトのコンセプトやビジョンを明確に定義する。2. ###利害関係者の特定 プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. ###要件の収集 利害関係者から要件を収集し、優先順位を付ける。4. ###要件の分析 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. ###優先順位の付与 要件に優先順位を与え、重要なものとそうでないものを区別する。6. ###要件仕様書の作成 収集された要件をまとめて、要件仕様書として文書化する。7. ###要件の確認と承認 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。</p>	<p>### 要件定義の方法 要件定義はプロジェクトの成功に不可欠なステップであり、以下の手順に従って行われます。1. ###ステークホルダーの特定 プロジェクトに関わる重要なステークホルダーを特定します。ステークホルダーには、利用者、顧客、開発チーム、マーケティング部門などが含まれます。2. ###情報収集 ステークホルダーから要件に関する情報を収集します。ヒアリング、ワークショップ、アンケートなどの手法が活用されます。3. ###要件の整理と分類 収集した情報を整理し、共通するテーマやパターンを特定します。要件を優先順位付けして分類することが重要です。4. ###要件文書の作成 要件を明確に文書化します。要件文書は、機能要件、非機能要件、制約要件などを含みます。5. ###要件の検証 ステークホルダーや関係者との間で要件を検証し、正確かつ完全なものであることを確認します。フィードバックを受け取り、必要に応じて修正を加えます。6. ###要件の承認 最終的な要件文書をステークホルダーに提示し、承認を取得します。要件の変更がある場合は、適切な手続きを経て変更を取り入れます。以上が、要件定義の一般的な手法の概要です。プロジェクトの性質や規模によって、要件定義手法は変化する場合があります。</p>	0	0.72902	0.80778	90	85	<p>両方の文章は要件定義のプロセスを説明しており、プロジェクトの成功に不可欠なステップであることを強調しています。ステークホルダーの特定、要件の収集、要件の文書化と承認といった共通のステップを含んでいます。入力文1はプロジェクトの背景と目的の明確化を含む一方で、入力文2は情報収集の手法や要件の分類について詳しく述べています。全体として、プロセスの流れや目的は非常に類似していますが、細部の焦点や説明の深さに若干の違いがあります。</p>

36	—	<p>### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区別する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。</p>	<p>### 1の要件定義パターン要件定義には、上記で紹介した一般的な手法以外にも、以下のような異なるパターンが存在します。1. **アジャイル開発における要件定義** - アジャイル開発では、要件定義を継続的なプロセスとして捉えます。一度に全ての要件を定義するのではなく、スプリントごとに必要な要件を追加・調整していきます。2. **ユースケース駆動開発 (Use Case Driven Development)** - ユースケースを中心に、要件を設定・定義します。システムの利用者やアクターが実際に行う活動に重点を置いた要件定義手法です。3. **プロトタイプング** - プロトタイプを作成し、利害関係者や利用者と共に試験・フィードバックを受けながら、要件を洗練させていく手法です。4. **ワークショップ** - 開発チームやステークホルダーとのワークショップを通じて、要件を共同で収集・定義します。チーム全体が参加することで、コミュニケーションを促進し共通理解を深めます。5. **顧客インタビュー** - 顧客や利用者との直接のインタビューを通じて、要件を明確にし、優先順位付けを行います。リアルなフィードバックを取り入れながら要件を洗練させていきます。これらは、要件定義を柔軟に行うためのいくつかの代替パターンです。プロジェクトの目標や状況に応じて、最適な要件定義方法を選択することが重要です。</p>	0	0.64687	0.58195	60	45	<p>両方の文章は要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は特定の要件定義プロセスを詳細に説明しており、プロジェクトの背景から要件の確認と承認までの流れを示しています。一方、入力文2は異なる要件定義の手法を紹介し、アジャイル開発やプロトタイプングなどの柔軟なアプローチを説明しています。したがって、要件定義というテーマは共通していますが、具体的な内容や焦点が異なるため、意味合い的な類似度は中程度です。</p>	<p>入力文1と入力文2は、どちらも要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は、伝統的な要件定義の手順を詳細に説明しており、プロジェクトの背景から要件の確認と承認までの一連のプロセスを順序立てて示しています。一方、入力文2は、アジャイル開発やユースケース駆動開発など、異なる要件定義のパターンを紹介しています。両者は要件定義という共通のテーマを持っていますが、具体的な手法やアプローチが異なるため、意味合い的な類似度は中程度と評価しました。</p>
37	—	<p>システム開発の振り返りを効果的に行うためには以下の手順が役立ちます：##1. **目標の確認** - 開発プロジェクトの目標が達成されたかどうかを確認します。##2. **プロジェクトの成功要因と課題の洗い出し** - 成功要因と現れた課題を特定し、分析します。##3. **フィードバックの収集** - 開発チームや関係者からフィードバックを収集し、意見を整理します。##4. **プロセスの評価** - 開発プロセスと運用を評価し、改善点を特定します。##5. **成果物の評価** - 開発されたソフトウェアや成果物の品質と適合性を確認します。##6. **振り返り会議の開催** - チーム全体で振り返り会議を開催し、経験を共有し、次のプロジェクトに活かせる教訓を引き出します。##7. **改善計画の策定** - 顕著な問題点を解決するための具体的な改善計画を立てます。##8. **成果の文書化** - 振り返り結果や改善計画を文書化し、関係者と共有します。これらのステップを踏むことで、今後のプロジェクトの成功につながる洞察を得ることができます。</p>	<p>### システム開発の振り返り方法システム開発のプロジェクトを振り返る際には、次のステップを考慮すると良いでしょう：1. **目標の確認** - まず、プロジェクトの始まりに設定された目標や要件を振り返りましょう。2. **成果物のレビュー** - 開発したシステムやソフトウェアを詳細に検討し、機能や品質について評価します。3. **チームのフィードバック** - プロジェクトに携わった全ての人からのフィードバックを収集し、プロセスやチームのパフォーマンスについて理解します。4. **課題の特定** - プロジェクト中に生じた問題や課題を明らかにし、今後の改善点として議論します。5. **成功要因の明確化** - プロジェクトの成功した部分や良かった点を特定し、将来のプロジェクトに活かすために考察します。6. **次のステップの計画** - 振り返りの結果を踏まえて、次のプロジェクトやフェーズに移る際のアクションプランを立てます。以上のステップを通じて、システム開発プロジェクトの振り返りを効果的に行うことができます。</p>	0	0.88022	0.88923	90	85	<p>両方の文章は、システム開発プロジェクトの振り返りを効果的に行うための方法を説明しています。目標の確認、フィードバックの収集、課題の特定、成功要因の明確化など、主要なテーマが共通しています。結論の表現や順序に違いはあるものの、伝えたい内容や目的はほぼ3です。</p>	<p>両方の文は、システム開発プロジェクトの振り返りに関する手順を説明しており、基本的なステップや目的が非常に類似しています。具体的には、目標の確認、成果物の評価、フィードバックの収集、課題の特定、成功要因の明確化、次のステップの計画といった要素が共通しています。ただし、入力文1はより詳細な手順を提供し、振り返り会議の開催や成果の文書化といった追加のステップが含まれているため、完全に一致しているわけではありません。これらの違いがスコアに反映されています。</p>

付録3.長文化セットの評価結果

- ・短文と長文のセットで1行に記載しています。
- ・長文から短文を引いた差を同様に記載しています。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1, 別=0で表しています。

短文							長文							差(長文-短文)			
NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度
66	66	1	0.95122	0.96673	95	95	132	66	1	0.99696	0.99153	98	95	0.04575	0.02479	3	0
67	67	0	0.88500	0.83577	60	70	133	67	0	0.99503	0.98875	85	95	0.11003	0.15297	25	25
68	68	1	0.84721	0.85831	90	85	134	68	1	0.99089	0.98809	98	98	0.14368	0.12978	8	13
69	69	0	0.91387	0.86809	40	60	135	69	0	0.97818	0.96897	90	95	0.06432	0.10089	50	35
70	70	0	0.93875	0.91830	70	85	136	70	0	0.99692	0.98479	90	95	0.05817	0.06648	20	10
72	72	0	0.87188	0.84732	90	75	137	72	0	0.98664	0.98259	90	95	0.11476	0.13528	0	20
73	73	0	0.82057	0.85983	90	85	138	73	0	0.95717	0.97115	95	90	0.13660	0.11131	5	5
111	111	1	0.91655	0.89746	100	95	139	111	1	0.97804	0.96826	100	95	0.06148	0.07080	0	0
112	112	1	0.96277	0.96710	95	95	140	112	1	0.97557	0.98119	98	95	0.01280	0.01409	3	0
113	113	0	0.90149	0.84192	75	70	141	113	0	0.95176	0.93341	90	85	0.05027	0.09149	15	15
114	114	1	0.85186	0.92515	95	95	142	114	1	0.98611	0.97416	100	95	0.13425	0.04901	5	0
115	115	1	0.95134	0.95962	95	95	143	115	1	0.99251	0.98502	98	95	0.04117	0.02541	3	0
116	116	0	0.94920	0.94555	70	70	144	116	0	0.99330	0.98345	90	95	0.04410	0.03789	20	25
117	117	1	0.97527	0.98402	98	95	145	117	1	0.99510	0.99397	98	98	0.01983	0.00994	0	3
118	118	1	0.95407	0.92963	95	95	146	118	1	0.98814	0.95558	98	95	0.03408	0.02595	3	0
119	119	0	0.98682	0.94501	60	85	147	119	0	0.99540	0.94425	80	95	0.00858	-0.00075	20	10
120	120	1	0.87244	0.84566	30	85	126	120	1	0.97904	0.94889	70	85	0.10660	0.10323	40	0
121	121	0	0.93091	0.87517	10	30	127	121	0	0.96875	0.93828	50	70	0.03784	0.06311	40	40
122	122	1	0.95623	0.96410	98	95	128	122	1	0.98664	0.98639	98	95	0.03041	0.02228	0	0
123	123	0	0.92170	0.88302	10	20	129	123	0	0.97480	0.90514	50	50	0.05310	0.02212	40	30
124	124	1	0.87822	0.93415	85	85	130	124	1	0.97862	0.96418	90	95	0.10040	0.03003	5	10
125	125	0	0.88555	0.86678	20	20	131	125	0	0.98770	0.97186	60	70	0.10215	0.10508	40	50

A. 実験環境

A.1 開発言語およびライブラリの選定

本研究では、以下の開発環境とライブラリを使用しています。

- **Python 3 系 (3.8 以降)**
- **openai** : OpenAI 社の埋め込みモデルや GPT モデルを利用
- **numpy** : ベクトル演算や類似度計算 (コサイン類似度)
- **scikit-learn** : ROC 曲線や AUC など機械学習関連の評価指標
- **matplotlib** : ROC 曲線の可視化

そのほか、以下のパッケージも必要に応じて使用します。

- **argparse** : コマンドライン引数処理
 - **configparser** : INI ファイル形式の設定ファイル読み込み
 - **csv** : CSV 入出力
-

B. 類似度評価ツール

B.1 概要

本ツールは、OpenAI の Embedding モデルを用いたテキスト類似度計算と、GPT モデルを用いた 2 つのテキストの比較・要約などの評価を自動的に行うためのスクリプトです。主な機能は以下のとおりです。

1. Embedding による類似度計算

- Embedding ベースで 2 つのテキストを数値ベクトル化し、コサイン類似度を算出

2. ChatGPT による評価

- GPT モデル (GPT-3.5 や GPT-4 など) に、2 つのテキストを含んだプロンプトを送り、その応答を取得

3. 単発評価または CSV 一括処理

- コマンドライン引数を使った単発評価
- CSV ファイルを入力し、複数テキストをまとめて評価した結果を CSV に出カ

B.2 主な構成要素

B.2.1 EmbeddingSimilarity クラス

- **役割:**
OpenAI Embedding API を呼び出してテキストを数値ベクトルに変換し、コサイン類似度を計算します。
- **主なメソッド:**

1. `get_embedding(text, model_name)`:
Embedding モデルを指定し、テキストを Embedding に変換。
2. `cosine_similarity(vec1, vec2)`:
ベクトル 2 つのコサイン類似度を計算。
3. `get_similarity(text1, text2, model_name)`:
2 つのテキストの Embedding を取得し、コサイン類似度を返す。

B.2.2 ChatGPTSimilarity クラス

- **役割:**
GPT (ChatGPT) を用いて 2 つのテキストを比較・評価する。
- **主なメソッド:**
 1. `get_chatgpt_evaluation(text1, text2, gpt_model, prompt)`:
指定した GPT モデルとプロンプトを使い、2 つのテキストに対する評価応答を取得する。

C. 使用方法

C.1 事前準備

C.1.1 Python 環境の準備

必要に応じて以下のライブラリをインストールしてください。

```
pip install openai numpy argparse configparser scikit-learn matplotlib pandas
```

※ `scikit-learn`, `matplotlib`, `pandas` は ROC 曲線ツールで使用します。

C.1.2 OpenAI API キーの取得

- [OpenAI アカウント](#)を作成し、API キーを発行しておきます。

※ APIは基本的に有料であり注意して使用してください。

C.1.3 設定ファイル (`config.ini`) の用意

同じディレクトリに `config.ini` を配置し、以下のように記述します。

```
[OpenAI]
api_key = sk-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

※ 実際の API キーはご自身のものに置き換えてください。

C.1.4 モデル定義ファイル (`models.json`) の用意

Embedding モデル一覧や GPT モデル+プロンプトの設定を JSON 形式で記述します。以下は今回実験で使った例です。

```
{
  "embedding_models": [
    "text-embedding-ada-002",
    "text-embedding-3-small",
    "text-embedding-3-large"
  ],
  "gpt_models_and_prompts": [
    {
      "gpt_model": "gpt-4o",
      "prompt": "#命令:以下の条件で入力文1と入力文2を比較して文章構造の類似度と意味合  

      いた類似度をそれぞれ評価してください。#制約条件:・文章構造の類似度と意味合いた類似度は1~100のスコア  

      で示すこと・それぞれの評価の根拠を簡潔に記載すること・入力文1と入力文2は出力しない #入力文1:  

      {text1} #入力文2:{text2} #出力文:スコア、根拠"
    },
    {
      "gpt_model": "gpt-4o",
      "prompt": "#命令:以下の条件で入力文1と入力文2を比較して意味合いた類似度をそれ  

      ぞれ評価してください。#制約条件:・意味合いた類似度は1~100のスコアで示すこと・評価の根拠を簡潔に記載  

      すること・入力文1と入力文2は出力しない #入力文1:{text1} #入力文2:{text2} #出力文:スコア、根拠"
    }
  ]
}
```

- `embedding_models`: 使用したい OpenAI の Embedding モデルを配列形式で指定
- `gpt_models_and_prompts`: 使用したい GPT モデルと、プロンプト文字列を指定。{text1} と {text2} の部分が実際に評価対象となるテキストに置換されて評価されます。

C.2 ディレクトリ構成例

以下のような形でファイルを配置することを想定しています（例）。

```
your_project/
├─ calculate_similarity_v4.py
├─ config.ini
├─ models.json
├─ input.csv
└─ output.csv
```

- `calculate_similarity_v4.py`
本ツールのメインスクリプト。
- `config.ini`
OpenAI API キーを記載した設定ファイル。
- `models.json`
使用する Embedding モデルや GPT モデル+プロンプトを定義したファイル。

- `input.csv`
CSV 一括評価で使用するテキストを記載するファイル (オプション)。
- `output.csv`
一括評価実行後の出力先ファイル (オプション)。

※ `input.csv` や `output.csv` は単発評価を行う場合には必須ではありません。

C.3 単発評価

C.3.1 コマンドライン引数で直接テキストを指定する場合

```
cd your_project
python calculate_similarity_v4.py \
  --text1 "今日は良い天気ですね" \
  --text2 "天候が素晴らしい1日です" \
  --config config.ini \
  --models_json models.json
```

- `models.json` に指定されたすべての Embedding モデルについて類似度を計算し、すべての GPT モデル+プロンプトについて評価を行います。
- 結果はコンソール上に表示されます。

C.4 CSV 一括評価

C.4.1 CSV ファイルの作成

1 行目にヘッダとして `text1,text2` を用意し、2 行目以降に評価対象のテキストを記述します。

```
text1,text2
"今日は良い天気ですね","天候が素晴らしい1日です"
"Pythonは便利です","プログラミング言語の中でも人気があります"
...
```

C.4.2 評価コマンド

```
cd your_project
python calculate_similarity_v4.py \
  --csv_input input.csv \
  --csv_output output.csv \
  --config config.ini \
  --models_json models.json
```

- `input.csv` 内のすべての行について、`text1` と `text2` のペアを評価します。
- 評価結果は `output.csv` に書き出されます。

C.4.3 出力 CSV のイメージ

`models.json` に書かれている複数のモデル設定に応じて、以下のような列が追記されることを想定しています。

- `text1`
- `text2`
- `Embedding_text-embedding-ada-002`
- `ChatGPT_gpt-3.5-turbo_次の2つの文章が...`
 - 上記プロンプトがヘッダに含まれるため、長くなる場合は適宜省略・整形を検討してください。

D. エラー処理

1. API エラー (`openai.OpenAIError`)

- API キーが誤っている場合や、OpenAI API に障害が発生している場合に起こります。プログラムは標準エラーへメッセージを出力します。

2. その他の予期しないエラー

- 文字コードが原因のファイル読み込みエラーや、CSV の列名が誤っているなどの場合も同様に標準エラーへ出力します。

E. ソースコード

```
import argparse
import csv
import sys
import configparser
import openai
import json
import numpy as np

class EmbeddingSimilarity:
    def __init__(self, api_key):
        # Initialize with the provided OpenAI API key
        openai.api_key = api_key

    def get_embedding(self, text, model_name):
        # Get the embedding for the given text using the specified model
        try:
            response = openai.embeddings.create(
                input=text,
                model=model_name
            )
            embedding = response.data[0].embedding
```

```
        return embedding
    except openai.OpenAIError as e:
        # Handle OpenAI-specific errors
        print(f"API Error: {e}", file=sys.stderr)
        raise e # 再入力
    except Exception as e:
        # Handle any other unexpected errors
        print(f"Unexpected Error: {e}", file=sys.stderr)
        raise e # 再入力

def cosine_similarity(self, vec1, vec2):
    # Convert vectors to numpy arrays for efficient computation
    vec1 = np.array(vec1)
    vec2 = np.array(vec2)

    # Compute the dot product of the two vectors
    dot = np.dot(vec1, vec2)

    # Compute the norms (magnitudes) of the vectors
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)

    # Return 0 if either vector has zero magnitude
    if norm1 == 0 or norm2 == 0:
        return 0.0

    # Return the cosine similarity (dot product divided by the product of the
norms)
    return dot / (norm1 * norm2)

def get_similarity(self, text1, text2, model_name):
    # Get embeddings for both texts and compute their cosine similarity
    emb1 = self.get_embedding(text1, model_name)
    emb2 = self.get_embedding(text2, model_name)
    return self.cosine_similarity(emb1, emb2)

class ChatGPTSimilarity:
    def __init__(self, api_key):
        # Initialize with the provided OpenAI API key
        openai.api_key = api_key

    def get_chatgpt_evaluation(self, text1, text2, gpt_model, prompt):
        try:
            formatted_prompt = prompt.format(text1=text1, text2=text2)
            response = openai.chat.completions.create(
                model=gpt_model,
                messages=[{"role": "user", "content": formatted_prompt}],
                temperature=0
            )
            return response.choices[0].message.content.strip()
        except Exception as e:
            print(f"API Error: {e}", file=sys.stderr)
            raise e # ここでも再入力しておくテストで例外検知しやすい
```

```
class TextSimilarityProcessor:
    def __init__(self, api_key, embedding_models, gpt_models_and_prompts):
        # Initialize with API key, embedding models, and GPT models with prompts
        self.embedding_models = embedding_models
        self.gpt_models_and_prompts = gpt_models_and_prompts
        self.api_key = api_key

    def process_pair(self, text1, text2):
        # Create instances for similarity calculations
        similarity_calculator = EmbeddingSimilarity(self.api_key)
        chatgpt_calculator = ChatGPTSimilarity(self.api_key)

        results = {}
        # Compute embedding-based similarities for each specified model
        for model in self.embedding_models:
            similarity = similarity_calculator.get_similarity(text1, text2, model)
            results[f"Embedding_{model}"] = similarity

        # Get ChatGPT-based evaluations for each specified model and prompt
        for entry in self.gpt_models_and_prompts:
            gpt_model = entry["gpt_model"]
            prompt = entry["prompt"].replace('\n', ' ').replace('\r', ' ')
            chatgpt_eval = chatgpt_calculator.get_chatgpt_evaluation(text1, text2,
gpt_model, entry["prompt"])
            # 結果キーをfieldnamesと同じ形式にする
            results[f"ChatGPT_{gpt_model}_{prompt}"] = chatgpt_eval
        return results

    def process_csv(input_file, output_file, api_key, embedding_models,
gpt_models_and_prompts):
        processor = TextSimilarityProcessor(api_key, embedding_models,
gpt_models_and_prompts)
        try:
            with open(input_file, newline='', encoding='utf-8') as csv_in, \
                open(output_file, 'w', newline='', encoding='utf-8') as csv_out:

                reader = csv.DictReader(csv_in)

                embedding_fields = [f"Embedding_{model}" for model in
embedding_models]
                chatgpt_fields = []
                for entry in gpt_models_and_prompts:
                    gpt_model = entry["gpt_model"]
                    prompt = entry["prompt"].replace('\n', ' ').replace('\r', ' ')
                    chatgpt_fields.append(f"ChatGPT_{gpt_model}_{prompt}")

                fieldnames = ['text1', 'text2'] + embedding_fields + chatgpt_fields
                writer = csv.DictWriter(csv_out, fieldnames=fieldnames)
                writer.writeheader()

                for row in reader:
                    try:
                        text1 = row['text1'].replace('\n', ' ').replace('\r', ' ')
                        text2 = row['text2'].replace('\n', ' ').replace('\r', ' ')
```

```
        results = processor.process_pair(text1, text2)
        row.update({k: (v.replace('\n', ' ').replace('\r', ' ') if
isinstance(v, str) else v) for k, v in results.items()})
        writer.writerow(row)
    except Exception as e:
        print(f"Error processing row: {e}", file=sys.stderr)
        raise e # 再度例外を発生させる

except Exception as e:
    print(f"Unexpected Error: {e}", file=sys.stderr)
    raise e

def main():
    # Parse command-line arguments
    parser = argparse.ArgumentParser(description="テキスト類似性評価ツール")
    parser.add_argument("--text1", type=str, help="評価テキスト1")
    parser.add_argument("--text2", type=str, help="評価テキスト2")
    parser.add_argument("--csv_input", type=str, help="CSV入力ファイル")
    parser.add_argument("--csv_output", type=str, help="CSV出力ファイル")
    parser.add_argument("--config", type=str, default="config.ini", help="設定ファイルパス")
    parser.add_argument("--models_json", type=str, help="EmbeddingモデルとGPTモデルおよびプロンプトが定義されたJSONファイルパス")

    args = parser.parse_args()

    # Ensure required arguments are provided
    if not (args.text1 and args.text2) and not (args.csv_input and
args.csv_output):
        print("Error: You must provide either --text1 and --text2 for single
evaluation, or --csv_input and --csv_output for batch processing.",
file=sys.stderr)
        parser.print_help()
        sys.exit(1)

    # Read API key from the configuration file
    config = configparser.ConfigParser()
    config.read(args.config, encoding='utf-8')
    api_key = config.get("OpenAI", "api_key", fallback=None)

    if api_key is None:
        print("OpenAI APIキーが指定されていません。", file=sys.stderr)
        sys.exit(1)

    # Load model configurations from the JSON file
    with open(args.models_json, "r", encoding="utf-8") as f:
        models_config = json.load(f)
        embedding_models = models_config.get("embedding_models", [])
        gpt_models_and_prompts = models_config.get("gpt_models_and_prompts", [])

    # Process based on the provided arguments
    if args.csv_input and args.csv_output:
        process_csv(args.csv_input, args.csv_output, api_key, embedding_models,
gpt_models_and_prompts)
```

```
elif args.text1 and args.text2:
    processor = TextSimilarityProcessor(api_key, embedding_models,
gpt_models_and_prompts)
    results = processor.process_pair(args.text1, args.text2)
    print("=== 評価結果 ===")
    for key, value in results.items():
        print(f"{key}: {value}")

if __name__ == "__main__":
    main()
```

F. ROC 曲線と AUC の算出ツール

本研究では、二値分類モデルの評価指標として ROC 曲線と AUC を算出するツールも併せて使用しました。このプログラムを使うと、以下のステップでモデルの ROC 曲線と AUC を計算・可視化できます。

F.3 ソースコード

```
import pandas as pd
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# 1. CSVファイルを読み込む
data = pd.read_csv('roc_input_data.csv') # CSVファイル名

# 2. True_Class と Predicted_Score を取得
y_true = data['True_Class']
y_score = data['Predicted_Score']

# 3. ROC曲線を計算
fpr, tpr, thresholds = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)

# 4. プロット
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve IMIAI BETSU')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

F.4 使用方法

1. ライブラリのインストール

必要に応じて以下をインストールします。

```
pip install pandas scikit-learn matplotlib
```

2. CSV ファイルの用意

- 列名を `Sample_ID, True_Class, Predicted_Score` として CSV を用意します。
- ファイル名は `roc_input_data.csv` としてコードと同じディレクトリに保存します。
- 例:

```
Sample_ID,True_Class,Predicted_Score
1,1,100
2,1,100
3,1,100
...
```

- `Sample_ID` はテストデータの番号を入力するための管理用の列です。アウトプットには影響しません。
- `True_Class` は二値 (0, 1) で設定。
- `Predicted_Score` は連続値で設定 (0.0~1.0 の範囲など)。

3. スクリプトの実行

上記コードを `.py` ファイルに保存し、ターミナルやコマンドプロンプトで `python your_script.py` を実行すると、ROC 曲線がプロットされ、AUC が図中の凡例に表示されます。

注意: 本プログラムは、`plt.show()` を用いてグラフを画面表示用ウィンドウとして描画します。そのため GUI 環境 (X ウィンドウシステムや macOS/Windows のデスクトップ環境など) が必要です。

G. その他注意事項

本ソースコードは学術研究・検証用であり、いかなる損害についても責任を負いません。商用利用や特定目的への適合性は保証しません。OpenAI API は有料です。大量のテキストを扱う場合は OpenAI API の料金に注意しつつ運用することをおすすめします。

付録2: 実験データと評価結果 (一部)

- ・NoはデータのユニークIDです。
- ・文脈的に合致した同じ文章をtext1とtext2の両方に挿入し、文章の長文化をしたもののセットを長文化セット列で表し値が一致するものがセットです。長文化種別は、元の短文か、長文化後の文章かを表しています。
- ・text1, text2は類似度を測る対象の文章です。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1, 別=0で表しています。
- ・text-embedding-3-smallから、ChatGPTスコアは各モデルでの類似度評価結果を表します。text-embedding系モデルの2種は0~1で表し、1に近いほど類似度が高いです。ChatGPT系モデルの2種は0~100で表し、100に近いほど類似度が高いです。
- ・ChatGPT系モデルは評価値の根拠も同時に出力したため、掲載しています。

No	長文化種別	長文化セット	text1	text2	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	ChatGPT比較類似度の根拠	ChatGPT独立類似度の根拠
1	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイル開発は反復的で柔軟性があり、変化に迅速に対応できるのに対し、ウォーターフォール開発は計画に基づいた直線的なプロセスで、各工程を順に完了させてから次に進む点が大きな違いです。	1	0.97910	0.98436	95	95	両方の文はアジャイル開発とウォーターフォール開発の違いを説明しており、意味合いはほぼ3です。どちらもアジャイル開発の柔軟性とウォーターフォール開発の計画重視を強調しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。	両方の文は、アジャイル開発とウォーターフォール開発の特徴と違いを説明しています。具体的には、アジャイル開発の反復性と柔軟性、変化への適応力を強調し、ウォーターフォール開発の計画重視と直線的なプロセスを指摘しています。表現の違いはあるものの、伝えている内容は非常に類似しています。
2	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイルは反復的で柔軟性が高く、変更対応を重視します。一方、ウォーターフォールは各工程を順序立てて進め、後戻りが難しい計画重視の手法です。	1	0.91004	0.89423	95	85	両方の文はアジャイル開発とウォーターフォール開発の特徴を説明し、それぞれの開発手法の違いを強調しています。アジャイルの「反復的」「柔軟性」「変更対応」とウォーターフォールの「計画重視」「順序立て」「後戻りが難しい」という点が共通しており、意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の特徴を比較しており、アジャイルの反復性と柔軟性、ウォーターフォールの計画重視と直線的な進行について言及しています。違いを強調する点で内容が非常に類似していますが、表現の仕方や細部に若干の違いがあります。
3	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	ウォーターフォールモデルは工程が順序立てられているため、柔軟性に欠け、要件変更や市場変化への対応が困難でした。この課題を解決するため、**反復的かつ適応的な開発**を重視するアジャイル開発が誕生しました。アジャイルは顧客との連携や短期間の反復（スプリント）を通じて、変化に対応しながら継続的に価値を提供する手法として進化しました。	0	0.76930	0.73989	85	85	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てており、特に柔軟性や適応性の違いを強調しています。文2はアジャイル開発の誕生背景を詳しく述べているため、文1よりも詳細ですが、基本的な意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てています。特に、アジャイル開発の柔軟性と反復性、ウォーターフォール開発の計画重視と柔軟性の欠如について述べています。両者ともにアジャイル開発がウォーターフォール開発の課題を解決するために生まれたことを示唆しており、基本的な意味合いは非常に類似しています。ただし、文2はアジャイル開発の進化や顧客との連携についても詳しく述べているため、若干の情報の違いがあります。
4	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：ユーザーは非対称暗号方式を用いて「公開鍵」と「秘密鍵」のペアを生成します。秘密鍵は安全に保管し、公開鍵は認証サーバや接続先に共有します。2. **公開鍵の登録**：ユーザーは公開鍵を接続先サーバに登録します。登録された公開鍵は、サーバ上の認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証の開始**：ユーザーがサーバに接続を試みると、サーバは認証の一環としてランダムなデータ（チャレンジ）をユーザーに送信します。4. **秘密鍵による署名**：ユーザーは受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵を用いて署名を検証します。このプロセスで、ユーザーが秘密鍵の正当な所有者であることが確認されます。6. **接続の許可**：認証が成功すると、サーバは接続を許可し、安全な通信が確立されます。この手順により、パスワードを使わずに高いセキュリティで認証が行われます。	1	0.97842	0.96964	98	95	両方の文章は、公開鍵認証の手順を説明しており、伝えたい内容や意味はほぼ同一です。細かい表現の違いはあるものの、全体としての意味合いに大きな差異はありません。	両方の文は公開鍵認証の手順について詳細に説明しており、手順の流れや内容が非常に類似しています。鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可といったステップが3順序で記載されています。違いは、用語の選択や表現のわずかな違いに過ぎず、全体の意味合いにはほとんど影響を与えません。したがって、意味合い的な類似度は非常に高いと評価できます。
5	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. ユーザーが鍵ペア（公開鍵と秘密鍵）を生成します。 2. 公開鍵をサーバーに登録します。 3. ユーザーが接続を試みると、サーバーは暗号化されたチャレンジを送信します。 4. ユーザーは秘密鍵でチャレンジを復号し、応答を返します。 5. サーバーが応答を確認し、認証を完了します。	0	0.89827	0.87419	90	85	両方の文章は公開鍵認証の基本的な手順を説明しており、意味合いとしては非常に類似しています。どちらも鍵ペアの生成、公開鍵の登録、チャレンジと応答のプロセスを通じて認証が行われることを述べています。違いは、入力文1がより詳細で技術的な説明を含んでいる点です。	両方の文は公開鍵認証の手順について説明しており、基本的な流れは非常に似ています。鍵ペアの生成、公開鍵の登録、チャレンジの送信と応答、そして認証の完了という主要なステップが共通しています。ただし、入力文1は手順をより詳細に説明しており、特に「署名」と「署名の検証」という用語を使用しているのに対し、入力文2は「復号」と「応答を返す」という表現を用いています。この違いが若干の意味合いの違いを生んでいますが、全体としては3プロセスを指しているため、高い類似度があります。
6	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. サーバーがユーザーに暗号化されたチャレンジを送信します。 2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバーに渡そうとします（実際は公開鍵を登録します）。 3. サーバーは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。 4. ユーザーが秘密鍵を使用して復号し、登録プロセスをやり直します。 5. 最終的に公開鍵がサーバーに登録され、接続が試みられます。	0	0.88345	0.83765	60	45	両方の文章は公開鍵認証のプロセスを説明していますが、具体的な手順や用語の使い方に違いがあります。入力文1は正確な手順を説明しているのに対し、入力文2は一部の手順が誤解を招く表現になっており、特に秘密鍵の取り扱いに関する記述が異なります。これにより、全体的な意味合いに違いが生じています。	両方の文は公開鍵認証の手順について説明していますが、手順の詳細や順序に違いがあります。入力文1は、鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可という具体的なステップを順序立てて説明しています。一方、入力文2は手順が曖昧で、特に秘密鍵の取り扱いに誤解を招く表現があります（秘密鍵をサーバーに渡そうとするなど）。また、手順の順序も異なり、鍵ペアの生成が認証プロセスの途中で促されるなど、実際の公開鍵認証の流れとは異なっています。したがって、意味合い的な類似度は中程度と評価しました。
7	—	—	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なるアプローチを採用しており、柔軟なスキーマ設計と高いスケラビリティを特徴とする非リレーショナルなデータベースです。	1	0.90452	0.89878	95	95	両方の文はNoSQLの特徴を説明しており、非リレーショナルデータベースであること、柔軟なスキーマ設計、スケラビリティの高さを強調しています。意味合いはほぼ3ですが、入力文2はリレーショナルデータベースとの対比を明示している点で若干の違いがあります。	両方の文は、NoSQLが非リレーショナルデータベースであること、柔軟なスキーマ設計を持つこと、高いスケラビリティを特徴とすることを述べています。文2は、リレーショナルデータベースとの違いを明示していますが、全体的な意味合いは非常に類似しています。

8	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なり、柔軟なデータモデルを採用したデータベースの総称です。スケラビリティや高速性に優れ、文書型、キー・値型、カラム型、グラフ型などの種類があります。	0	0.83348	0.82690	85	85	両方の文はNoSQLの特徴として「柔軟性」と「スケラビリティ」を挙げており、NoSQLがリレーショナルデータベースとは異なることを強調しています。入力文2はさらに「高速性」や具体的なデータモデルの種類についても触れていますが、全体的な意味合いは非常に類似しています。	両方の文はNoSQLデータベースについて述べており、非リレーショナルであること、柔軟なデータモデルやスキーマ設計を持つこと、スケラビリティに優れていることを共通して説明しています。入力文2はさらにNoSQLの種類や高速性についても言及していますが、基本的な特徴に関する情報は一致しているため、高い類似度があります。
9	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	SQLに「No」を突き付ける人々は、スケラビリティや柔軟性を重視し、NoSQL(非リレーショナルデータベース)を支持する傾向があります。彼らは、大量データ処理や動的スキーマ、水平分散が必要なユースケース(例:リアルタイム分析や分散型アプリ)でSQLの制約を問題視します。MongoDBやCassandraなどのNoSQLは、特定の用途でSQLを補完・代替する選択肢として注目されています。	0	0.71857	0.74304	75	85	両方の文はNoSQLの特徴である「スケラビリティ」や「柔軟性」に言及しており、NoSQLが非リレーショナルデータベースであることを共通して述べています。ただし、入力文2はNoSQLを支持する理由や具体的なユースケースについても詳しく述べているため、意味合い的にはより広範な内容を含んでいますが、基本的なテーマは一致しています。	両方の文はNoSQLデータベースに関する内容であり、特にスケラビリティや柔軟なスキーマ設計の利点を強調しています。入力文1はNoSQLの特徴を簡潔に述べており、入力文2はNoSQLを支持する理由や具体的なユースケースを詳しく説明しています。内容の焦点が一致しており、NoSQLの利点に関する理解が共通しているため、高い類似度があります。ただし、入力文2はSQLとの比較や具体例を含むため、詳細度に差があります。
10	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	要件定義や設計段階での問題が原因である可能性が高いです。十分に練られていない要件定義や設計上のミスが、テスト工程における障害の多発に直結します。	1	0.92471	0.91897	95	95	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であるという3意味を伝えています。表現の違いはあるものの、伝えたい内容はほぼ同一です。	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であることを述べています。言い回しや表現に若干の違いはありますが、伝えたい内容や論点は非常に類似しています。
11	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	ウォーターフォール開発でシステムテスト工程で障害が多発する場合、主に**要件定義**や**設計**に問題がある可能性が高いです。要件の不備や曖昧さが原因で正しい仕様が実装されていない場合や、設計段階でのミスがテスト工程で顕在化します。また、**結合テスト**での検証不足や、単体テストでの抜け漏れも要因となり得ます。	0	0.75388	0.80472	85	85	両方の文は、要件定義や設計の不備がテスト工程での障害多発の原因であることを示しています。入力文2はより具体的な開発プロセス(ウォーターフォール開発)やテスト工程の詳細を含んでいますが、基本的な意味合いは一致しています。	両方の文は、要件定義や設計工程に問題があるとテストで障害が多発する可能性が高いことを述べています。入力文1は要件定義や設計ミスが直接テストでの障害につながると述べており、入力文2はウォーターフォール開発におけるシステムテストでの障害多発の原因として要件定義や設計の問題を挙げています。入力文2はさらに結合テストや単体テストの不足も要因として挙げており、詳細な説明が加えられていますが、基本的な意味合いは非常に類似しています。
12	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	システムで障害が多発するケースには、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障が挙げられます。特にスケラビリティを考慮しない設計や負荷テスト不足は、アクセス急増時に問題を引き起こします。また、運用プロセスの不備や変更管理の失敗、冗長性の欠如も影響します。継続的な監視や定期的なメンテナンスが重要です。	0	0.65997	0.53354	60	60	両方の文はシステムやプロジェクトにおける問題の原因について述べていますが、入力文1は要件定義や設計ミスに特化しており、入力文2はより広範な原因をカバーしています。共通点として設計の不備が挙げられますが、入力文2は他の要因も含むため、意味合い的な類似度は中程度です。	両方の文はシステムにおける障害の原因について述べていますが、焦点が異なります。入力文1は主に要件定義や設計工程の問題に焦点を当てており、これがテストでの障害多発につながるとしています。一方、入力文2はシステム障害の原因をより広範に捉え、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障などを挙げています。特にスケラビリティや運用プロセスの不備についても言及しています。共通点として設計の問題が挙げられますが、入力文2はより多くの要因を考慮しており、具体的な対策についても触れています。したがって、類似度は中程度と評価しました。
13	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社で物理サーバーを所有し運用する形式であり、一方クラウドはインターネットを通じて必要なリソースを利用する形態です。	1	0.95975	0.95273	95	95	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、内容的に非常に近いです。どちらもオンプレミスが自社でのサーバー管理を指し、クラウドがインターネット経由でのリソース利用を指しているため、伝えたい内容はほぼ同一です。表現の違いはあるものの、意味の違いはほとんどありません。	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、内容的に非常に類似しています。どちらもオンプレミスが自社で物理サーバーを所有・管理する形式であること、クラウドがインターネット経由でリソースを利用する形式であることを述べています。言い回しに若干の違いはありますが、意味合いはほぼ3です。
14	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社でサーバーやインフラを構築・管理する方式で、初期投資や保守が必要です。一方、クラウドはインターネット経由で外部サービスを利用する方式で、スケラビリティやコスト効率に優れます。	0	0.82750	0.76366	85	85	両方の文はオンプレミスとクラウドの違いを説明しており、基本的な意味合いは非常に近いです。オンプレミスは自社での管理が必要であること、クラウドはインターネット経由で利用することを述べています。入力文2は追加でスケラビリティやコスト効率についても言及していますが、全体的な意味合いは一致しています。	両方の文は、オンプレミスとクラウドの違いについて説明しています。オンプレミスについては、自社でサーバーを保有・管理する点が共通しており、クラウドについてはインターネット経由でリソースを利用する点が一致しています。違いとしては、入力文2が初期投資や保守、スケラビリティ、コスト効率についても言及しているため、詳細度に差がありますが、基本的な意味合いは非常に類似しています。
15	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	クラウド移行は、スケラビリティ、コスト効率、災害対策の強化、運用負荷の軽減を実現します。リソースを柔軟に拡張でき、初期投資を抑えつつ、セキュリティや最新技術の恩恵を享受可能です。競争力向上に寄与します。	0	0.44651	0.34687	40	30	両方の文はクラウドに関連していますが、入力文1はオンプレミスとクラウドの基本的な違いを説明しているのに対し、入力文2はクラウド移行の具体的な利点に焦点を当てています。意味的には関連性がありますが、焦点が異なるため、意味合い的な類似度は中程度です。	入力文1はオンプレミスとクラウドの基本的な違いを説明しており、主に技術的な形態に焦点を当てています。一方、入力文2はクラウド移行の利点について述べており、スケラビリティやコスト効率、競争力向上などの具体的なメリットに焦点を当てています。両者はクラウドに関連する内容を含んでいるものの、焦点が異なるため、意味合い的な類似度は低いです。
16	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデータを体系的に管理するシステム全体を指し、複数のテーブルを含む容器のようなものです。一方、テーブルはデータベース内でデータを行と列で構造化して格納する単位です。	1	0.89072	0.87697	95	95	両方の文はデータベースとテーブルの関係を説明しており、意味合いは非常に似ています。どちらもデータベースをデータを管理するシステム全体として説明し、テーブルをデータを構造化して保存する単位として説明しています。入力文2の方が若干詳細ですが、基本的な意味合いは一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であること、テーブルがデータを行と列で構造化して保存する単位であることを述べています。文2は、データベースを「複数のテーブルを含む容器」と表現しており、文1よりも少し詳細ですが、基本的な意味合いは非常に類似しています。
17	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースは、データを体系的に管理・保存するための**全体の容器**や構造を指します。一方、テーブルはデータベース内でデータを格納するための**個々の構造化された単位**です。データベースは複数のテーブルを持つことができ、それぞれのテーブルが行と列を持ち、特定の種類のデータを扱います。データベースはシステム全体、テーブルはその中の部品と考えられます。	0	0.86902	0.82861	85	95	両方の文はデータベースとテーブルの役割と関係性を説明しており、基本的な意味合いは一致しています。入力文2はより詳細で具体的な説明を提供していますが、全体的な意味は入力文1と一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であること、テーブルがその中でデータを構造化して保存する単位であることを述べています。入力文2は、入力文1の内容をより詳細に説明しており、データベースが複数のテーブルを持つことや、テーブルが行と列を持つことについても触れています。全体として、両方の文は3概念を伝えており、意味合い的に非常に類似しています。

18	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデジタル情報を整理・管理するシステムで、データを効率的に保存・検索・操作できます。一方、テーブルは物理的な家具で、物を置く・作業する場を提供します。データベースは抽象的で機能的、テーブルは具体的に物理的な用途に特化しています。	0	0.79866	0.67894	40	40	両方の文は「データベース」と「テーブル」に関する情報を提供していますが、文1はデータベース内のテーブルの役割を説明しているのに対し、文2はデータベースとテーブルを異なる文脈で比較しています。文2では、データベースは情報管理システムとしての機能を強調し、テーブルは物理的な家具としての役割を説明しており、意味合い的には異なる内容を持っています。	両方の文は「データベース」と「テーブル」という用語を扱っていますが、それぞれの文での「テーブル」の意味が異なります。入力文1では、データベース内の構造化されたデータの保存単位としての「テーブル」を説明していますが、入力文2では物理的な家具としての「テーブル」を説明しています。データベースに関する説明は類似していますが、テーブルに関する説明が異なるため、全体としての意味合い的類似度は低くなります。
19	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合するテーブルの両方で一致する行のみを取得します。一方、**OUTER JOIN**は一致する行に加えて、一方のテーブルにしか存在しない行も含めることができます。	1	0.85767	0.87835	95	95	両方の文はINNERJOINとOUTERJOINの機能の違いを説明しており、基本的な意味合いは非常に近いです。INNERJOINが両テーブルで一致するデータを取得し、OUTERJOINが片方のテーブルに一致しないデータも含めるという点で、意味の一致が見られます。表現の違いはあるものの、伝えたい内容はほぼ3です。	両方の文は、データベースにおけるINNER JOINとOUTER JOINの機能を説明しています。内容としては、INNER JOINが両方のテーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルにしか存在しないデータも含めることを述べています。言い回しや細かい表現の違いはあるものの、伝えている情報はほぼ3であるため、高い類似度スコアを付けました。
20	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合条件を満たす両方のテーブルの共通部分のみを取得します。一致しないデータは結果に含まれません。 **OUTER JOIN**は、条件を満たさないデータも含める結合で、以下の種類があります： - **LEFT OUTER JOIN**：左側のテーブルの全データと一致する右側のデータ。 - **RIGHT OUTER JOIN**：右側のテーブルの全データと一致する左側のデータ。 - **FULL OUTER JOIN**：両方のテーブルの全データを取得します。	0	0.81617	0.80728	85	85	両方の文はINNERJOINとOUTERJOINの基本的な概念を説明しており、意味合いは非常に似ています。ただし、入力文2はOUTERJOINの種類についての追加情報を提供しているため、完全に同一の意味合いではありません。	両方の文は、INNER JOINとOUTER JOINの基本的な概念を説明しており、特にINNER JOINについては非常に類似した内容を述べています。OUTER JOINに関しては、入力文2がLEFT、RIGHT、FULL OUTER JOINの詳細を追加しているため、情報量が多く、やや異なる視点を提供していますが、基本的な意味合いは一致しています。したがって、全体的な意味合い的類似度は高いと評価できます。
21	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	INNER JOINをOUTER JOINに書き換えるには、結合タイプを明示的に指定します。INNER JOINでは共通データのみ取得しますが、OUTER JOINでは片方のテーブルの全行を保持します。	0	0.82297	0.72906	80	85	両方の文はINNERJOINとOUTERJOINの違いについて説明していますが、入力文2は書き換え方法についても触れており、意味合いに若干の追加情報があります。それでも、基本的な意味合いは非常に似ています。	両方の文はINNER JOINとOUTER JOINの違いについて説明しています。入力文1は、INNER JOINが両テーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルに一致しないデータも含めることを述べています。入力文2も同様に、INNER JOINが共通データのみを取得し、OUTER JOINが片方のテーブルの全行を保持することを説明しています。ただし、入力文2はINNER JOINをOUTER JOINに書き換える方法についても触れており、結合タイプの指定についての情報が追加されています。このため、内容の焦点が若干異なる部分があるものの、基本的な意味合いは非常に類似しています。
22	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	用途によりますが、クラウドは初期コストが抑えられ柔軟性が高い一方で、長期的には利用状況に応じてコストが増加する可能性があります。	1	0.88793	0.93630	95	95	両方の文はクラウドの利点と欠点を述べており、初期コストの低さと柔軟性の高さを利点として挙げ、長期的なコスト増加の可能性を欠点として指摘しています。意味合いはほぼ3ですが、文2では「可能性があります」と若干の不確実性を示しています。	両方の文は、クラウドの特性について述べており、初期コストの低さと柔軟性の高さ、そして長期的なコスト増加の可能性について言及しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。
23	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	長期的なコスト効率は利用状況によりますが、一般的に次のように考えられます：- **クラウド**：初期投資が少なく、使用量に応じた課金モデルのため、柔軟性が高くスケラブルな運用が可能。ただし、長期で大規模な利用ではランニングコストが増える場合も。- **オンプレミス**：初期投資が大きい一方、長期間にわたって一定規模の利用を続ける場合、運用コストが安定し、総コストが抑えられることがあります。ただし、保守・更新コストが追加される点に留意。必要な規模や用途によってどちらが効率的かが異なります。	0	0.78180	0.72671	75	85	両方の文はクラウドの初期コストの低さと長期的なコスト増加の可能性について述べています。意味合いとしては、クラウドのコストに関する利点と欠点を説明している点で類似していますが、入力文2はオンプレミスとの比較も含んでおり、より広範な視点を提供しています。	両方の文はクラウドのコストと柔軟性に関する内容を扱っており、特に初期コストの低さと利用量に応じたコスト増加について言及しています。入力文2はオンプレミスについても触れていますが、クラウドに関する部分は入力文1と非常に類似しています。したがって、意味合い的な類似度は高いと評価できます。
24	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	オンプレミスは、一度の設備投資で長期間安定稼働が可能で、クラウドのような毎月のランニングコストが発生しません。大容量データや高負荷のアプリケーションでも、自社内で管理することで追加コストを抑えられます。特に既存設備の活用やカスタマイズ性が求められる環境では、オンプレミスが最も合理的で経済的な選択です！	0	0.56646	0.58247	50	65	両方の文章はITインフラの選択肢に関する情報を提供していますが、焦点が異なります。入力文1はクラウドの特性に焦点を当てており、入力文2はオンプレミスの利点を強調しています。どちらもコストと柔軟性に関する情報を含んでいますが、比較の対象が異なるため、意味合い的には部分的にしか重なりません。	両方の文はITインフラの選択肢であるクラウドとオンプレミスの特徴を説明しています。入力文1はクラウドの利点とコストの増加について述べており、入力文2はオンプレミスの利点とコスト管理について述べています。どちらもコストと柔軟性に関する情報を提供しており、ITインフラの選択に関する議論の一部を形成しています。ただし、具体的な内容や視点が異なるため、完全に一致するわけではありません。
25	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は、工程を順番に進める直線的な開発モデルです。主な流れは以下の通りです： 1. **要件定義**：システムの要件や仕様を明確に定義します。 2. **設計**：システム全体の構造を設計し、詳細な設計も行います。 3. **実装**：設計内容に基づいてプログラムを開発します。 4. **テスト**：システムが要件を満たしているかを確認します。 5. **運用と保守**：システムをリリースし、運用や問題修正を行います。 各工程が完了してから次に進む仕組みのため、途中で工程を見直す際には大きな手間がかかります。	1	0.98439	0.97103	98	95	両方の文章はウォーターフォール開発の特徴と工程を説明しており、意味合いに大きな違いはありません。用語の若干の違い(例:「要求定義」vs「要件定義」)や表現の微細な違いはありますが、伝えたい内容はほぼ3です。	両方の文はウォーターフォール開発モデルについて説明しており、工程の順序や内容がほぼ3です。用語の違い(「要求定義」vs「要件定義」など)や表現のわずかな違いはありますが、全体的な意味合いは非常に類似しています。工程の進め方や見直しの手間についても同様の説明がされています。
26	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進みます：1. **要件定義**：システムの要件や仕様を明確化する。2. **基本設計**：システム全体の構造や外部仕様を設計する。3. **詳細設計**：プログラム単位の具体的な設計を行う。4. **実装(コーディング)**：設計に基づきプログラムを作成する。5. **単体テスト**：個1のプログラムやモジュールをテストする。6. **結合テスト**：モジュール同士を結合して動作を確認する。7. **システムテスト**：システム全体の動作を確認し、要件通りか検証する。8. **運用・保守**：完成したシステムを運用し、問題が発生した場合に修正する。各工程が順に完了しないと次に進めないのが特徴です。	0	0.93794	0.93866	90	85	両方の文章はウォーターフォール開発の基本的な考え方と工程の流れを説明しており、意味合いは非常に似ています。どちらも工程を順序立てて進めることの重要性を強調していますが、入力文2は工程をより詳細に説明しているため、若干の違いがあります。	両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要求定義から運用・保守までの流れを説明しており、全体的なプロセスの流れは非常に似ています。ただし、入力文2はより詳細な工程(基本設計、詳細設計、単体テスト、結合テスト、システムテスト)を含んでおり、工程の細分化がされています。この違いがスコアに影響していますが、全体的な意味合いは非常に類似しています。

27	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進行します：5. **単体テスト**：個1のプログラムやモジュールをテストする。 1. **要件定義**：システムの要件や仕様を明確化する。 7. **システムテスト**：システム全体の動作を確認し、要件通りか検証する。 3. **詳細設計**：プログラム単位の具体的な設計を行う。 8. **運用・保守**：完成したシステムを運用し、問題が発生した場合に修正する。 4. **実装（コーディング）**：設計に基づきプログラムを作成する。 6. **結合テスト**：モジュール同士を結合して動作を確認する。 2. **基本設計**：システム全体の構造や外部仕様を設計する。 各工程が順に完了しないと次に進めないのが特徴です。	0	0.91345	0.91544	90	85	両方の文はウォーターフォール開発モデルの基本的な概念とその工程について説明しています。両方も、工程が順序立てて進行し、各工程が完了しないと次に進めないという特徴を強調しています。意味合い的には非常に類似しています。が、入力文2はテスト工程をより詳細に説明しており、若干の違いがあります。それでも、全体的な意味合いはほぼ3です。	両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要件定義、設計、実装、テスト、運用・保守といった主要な工程を含んでおり、全体的な流れも似ています。ただし、入力文2はテスト工程をより詳細に分けて説明しており、工程の順序が異なる部分もありますが、このため、完全に一致するわけではありませんが、基本的な概念と流れは非常に類似しています。
28	-	-	Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**：`javac`コンパイラを使用してソースコードをコンパイルし、バイトコード（`.class`ファイル）に変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムの実行手順は以下の通りです：1. **ソースコード作成**：Javaでプログラムを記述し、`java`拡張子のファイルとして保存します。 2. **コンパイル**：`javac`コンパイラを使ってソースコードをバイトコード（`.class`ファイル）に変換します。 3. **クラスローダーによるロード**：JVM（Java仮想マシン）がバイトコードを読み込み、メモリ上に展開します。 4. **バイトコードの実行**：JVM内でインタプリタまたはJIT（Just-In-Time）コンパイラがバイトコードをネイティブコードに変換し、実行します。 5. **結果の表示**：プログラムの実行結果が出力されます。 このプロセスにより、Javaはプラットフォームに依存せず動作する特性を持っています。	1	0.90186	0.95154	98	95	両方の文は、Javaプログラムの実行手順を説明しており、内容的にはほぼ同一です。細かい表現の違い（例：“動作順序”vs“実行手順”、“結果表示”vs“結果の表示”）はありますが、全体の意味合いに大きな違いはありません。	両方の文は、Javaプログラムの実行手順について詳細に説明しており、内容的にはほぼ同一です。手順の順序や内容も一致しており、用語の違い（例えば「動作順序」と「実行手順」）や表現のわずかな違い（例えば「結果表示」と「結果の表示」）があるものの、意味合いに大きな違いはありません。したがって、意味合い的な類似度は非常に高いと評価できます。
29	-	-	Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**：`javac`コンパイラを使用してソースコードをコンパイルし、バイトコード（`.class`ファイル）に変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムが動作する基本的な順序は以下の通りです：1. **ソースコード作成** Javaファイル（`.java`）にソースコードを記述します。2. **コンパイル** `javac`コンパイラを使用して、ソースコードをバイトコード（`.class`ファイル）に変換します。 例：`javac MyProgram.java` 3. **クラスローダーによるロード** Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。4. **バイトコードの検証** JVMのバイトコードベリファイアがコードの安全性を確認します。5. **実行準備** JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。6. **実行（メインメソッドの呼び出し）** 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例：`java MyProgram` 7. **コード実行** JVMがバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。8. **終了** プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。	0	0.92306	0.92074	90	85	両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。どちらもJavaのプラットフォーム非依存性を強調し、ソースコードの作成から実行までの流れを説明しています。入力文2はより詳細な説明を含んでいますが、全体的な意味合いは一致しています。	両方の入力文は、Javaプログラムの動作順序について説明しており、基本的な流れは非常に類似しています。具体的には、ソースコードの作成、コンパイル、クラスローダーによるロード、バイトコードの実行といった主要なステップが共通しています。ただし、入力文2にはバイトコードの検証や実行準備、プログラム終了時のメモリ解放といった詳細なステップが追加されており、より詳細なプロセスが記述されています。このため、全体的な流れは似ているものの、細部において差異があるため、完全に同一の意味合いとは言えません。
30	-	-	Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**：`javac`コンパイラを使用してソースコードをコンパイルし、バイトコード（`.class`ファイル）に変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムが動作する基本的な順序は以下の通りです：1. **実行準備** JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。 2. **ソースコード作成** Javaファイル（`.java`）にソースコードを記述します。 3. **終了** プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。 4. **クラスローダーによるロード** Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。 5. **コンパイル** `javac`コンパイラを使用して、ソースコードをバイトコード（`.class`ファイル）に変換します。 例：`javac MyProgram.java` 6. **バイトコードの検証** JVMのバイトコードベリファイアがコードの安全性を確認します。 7. **コード実行** JVMがバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。 8. **実行（メインメソッドの呼び出し）** 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例：`java MyProgram` この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。	0	0.89679	0.87155	80	75	両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。どちらもJavaがプラットフォームに依存せずに動作することを強調しています。ただし、入力文2はより詳細な手順を含んでおり、特定のプロセス（例：バイトコードの検証、メインメソッドの呼び出し）についての追加情報を提供しています。これにより、意味合いの細部に若干の違いがありますが、全体的な意味合いは一致しています。	両方の文はJavaプログラムの動作順序について説明していますが、詳細なステップや順序の違いがあります。入力文1は、ソースコード作成から結果表示までの基本的な流れを簡潔に説明しています。一方、入力文2は、実行準備やバイトコードの検証、メインメソッドの呼び出しなど、より詳細なステップを含んでいます。両方の文がJavaのプラットフォーム非依存性に言及している点は共通していますが、具体的な手順の記述に違いがあるため、完全に一致するわけではありません。したがって、意味合い的な類似度は75と評価しました。
31	-	-	ウォーターフォール開発は以下の順序で進行します：1. **要件定義**：システムの要件を明確化。 2. **基本設計**：全体の構造を設計。 3. **詳細設計**：プログラム単位の具体設計。 4. **実装**：設計に基づきプログラム作成。 5. **単体テスト**：各プログラムをテスト。 6. **結合テスト**：モジュール間の動作確認。 7. **システムテスト**：システム全体を検証。 8. **運用・保守**：問題修正や改善を行う。順序を厳密に守ることが特徴です。	ウォーターフォール開発は以下の順序で進行します：3. **詳細設計**：プログラム単位の具体設計。 7. **システムテスト**：システム全体を検証。 1. **要件定義**：システムの要件を明確化。 6. **結合テスト**：モジュール間の動作確認。 4. **実装**：設計に基づきプログラム作成。 8. **運用・保守**：問題修正や改善を行う。 2. **基本設計**：全体の構造を設計。 5. **単体テスト**：各プログラムをテスト。 順序を厳密に守ることが特徴です。	0	0.96852	0.95236	95	85	両方の文はウォーターフォール開発のプロセスとその特徴である「順序を厳密に守ること」を説明しています。ほぼ3です。	両方の文はウォーターフォール開発のプロセスを説明しており、3ステップを含んでいます。しかし、入力文2ではステップの順序が異なっており、ウォーターフォール開発の特徴である「順序を厳密に守ること」が反映されていません。このため、意味合い的には類似していますが、順序の重要性が異なるため完全に一致しません。
32	-	-	Javaプログラムの動作順序は以下の通りです：1. **ソースコード作成**：Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**：`javac`コンパイラを使用してソースコードをコンパイルし、バイトコード（`.class`ファイル）に変換。3. **クラスローダーによる読み込み**：JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**：JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**：プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。	Javaプログラムが動作する基本的な順序は以下の通りです：1. **実行準備** JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。 2. **ソースコード作成** Javaファイル（`.java`）にソースコードを記述します。 3. **終了** プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。 4. **クラスローダーによるロード** Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。 5. **コンパイル** `javac`コンパイラを使用して、ソースコードをバイトコード（`.class`ファイル）に変換します。 例：`javac MyProgram.java` 6. **バイトコードの検証** JVMのバイトコードベリファイアがコードの安全性を確認します。 7. **コード実行** JVMがバイトコードを解釈（またはJITコンパイル）し、各命令をCPUで実行します。 8. **実行（メインメソッドの呼び出し）** 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例：`java MyProgram` この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。	0	0.89726	0.87175	80	75	両方の文章はJavaプログラムの実行プロセスを説明しており、基本的な意味合いは3です。どちらもJavaのプラットフォーム非依存性を強調しています。ただし、入力文2はより詳細な手順を含んでおり、特に「実行準備」や「バイトコードの検証」など、入力文1にはない具体的なプロセスが追加されています。これにより、意味合いに若干の違いが生じていますが、全体としては3テーマを扱っています。	両方の文はJavaプログラムの動作順序について説明していますが、詳細なステップや順序の違いがあります。入力文1は、ソースコード作成から結果表示までの基本的な流れを簡潔に説明しています。一方、入力文2は、実行準備やバイトコードの検証、メインメソッドの呼び出しなど、より詳細なステップを含んでいます。両方の文がJavaのプラットフォーム非依存性に言及している点は共通していますが、具体的なプロセスの説明に違いがあるため、完全に一致するわけではありません。したがって、意味合い的な類似度は75と評価しました。

33	—	—	公開鍵認証の手順は以下の通りです: 1. **鍵ペアの生成** : 利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録** : 利用者は公開鍵を接続サーバ(認証サーバなど)に登録します。公開鍵はサーバの認証ファイル(例: ~/.ssh/authorized_keys)に保存されます。3. **認証プロセスの開始** : 利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ(チャレンジ)を送信します。4. **秘密鍵で署名** : 利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証** : サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可** : 認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです: 1. サーバがユーザーに暗号化されたチャレンジを送信します。 2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバに渡そうとします(実際は公開鍵を登録します)。 3. サーバは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。 4. ユーザーが秘密鍵を使用して復号し、登録プロセスをやり直します。 5. 最終的に公開鍵がサーバに登録され、接続が試みられます。	0	0.88341	0.83840	50	45	両方の文章は公開鍵認証のプロセスを説明しようとしていますが、入力文2には誤解を招く表現や不正確な情報が含まれています(例:秘密鍵をサーバに渡そうとするなど)。そのため、意味合い的には一致していない部分が多く、類似度は低くなります。	両方の文は公開鍵認証の手順について説明していますが、手順の詳細や順序に違いがあります。入力文1は、鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可という具体的なステップを順序立てて説明しています。一方、入力文2は手順が曖昧で、秘密鍵をサーバに渡そうとするなど、実際のプロセスとは異なる記述があります。全体として、両方の文は公開鍵認証に関連していますが、具体的な手順や正確さにおいて大きな違いがあります。
34	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区別する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 要件定義の流れ1. **プロジェクトの背景理解** - プロジェクトの目的、スコープ、ステークホルダーを理解する。2. **要件収集** - ステークホルダーとのコミュニケーションを通じて、必要な機能や要件を収集する。3. **要件分析** - 収集した要件を整理し、矛盾点や抜け漏れを特定する。4. **要件の整理と整合性の確認** - 複数の要件を整理し、矛盾や重複を解消して要求事項を洗練させる。5. **要件文書の作成** - 要求事項を記述した要件仕様書を作成し、関係者と共有する。6. **要件の検証** - 作成された要件がプロジェクトの目的やスコープと整合しているかを確認する。7. **フィードバックと調整** - ステークホルダーからのフィードバックを受け、必要に応じて要件を調整する。8. **最終的な要件の承認** - 最終的な要件がすべてのステークホルダーによって承認される。	1	0.81901	0.91259	90	85	両方の文章は、要件定義のプロセスを説明しており、プロジェクトの背景理解、要件の収集と分析、文書化、確認と承認といった主要な目的や意味合いが非常に似ています。細部の表現やステークホルダーとの関係の強調に若干の違いがありますが、全体的な意味合いはほぼ一致しています。	両方の文は要件定義のプロセスを説明しており、全体的な流れやステップが非常に類似しています。具体的には、プロジェクトの背景理解、要件の収集と分析、要件仕様書の作成、要件の確認と承認といった主要なステップが共通しています。ただし、入力文2には「フィードバックと調整」というステップが追加されており、要件の整合性確認やステークホルダーとのコミュニケーションに重点が置かれています。一方、入力文1は優先順位付けに関する詳細が含まれています。これらの違いがあるため、完全に同一の内容ではありませんが、全体的なプロセスの流れは非常に似ています。
35	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区別する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 要件定義の方法要件定義はプロジェクトの成功に不可欠なステップであり、以下の手順に従って行われます。1. **ステークホルダーの特定** : プロジェクトに関わる重要なステークホルダーを特定します。ステークホルダーには、利用者、顧客、開発チーム、マーケティング部門などが含まれます。2. **情報収集** : ステークホルダーから要件に関する情報を収集します。ヒアリング、ワークショップ、アンケートなどの手法が活用されます。3. **要件の整理と分類** : 収集した情報を整理し、共通するテーマやパターンを特定します。要件を優先順位付けして分類することが重要です。4. **要件文書の作成** : 要件を明確に文書化します。要件文書は、機能要件、非機能要件、制約要件などを含みます。5. **要件の検証** : ステークホルダーや関係者との間で要件を検証し、正確かつ完全なものであることを確認します。フィードバックを受け取り、必要に応じて修正を加えます。6. **要件の承認** : 最終的な要件文書をステークホルダーに提示し、承認を取得します。要件の変更がある場合は、適切な手続きを経て変更を取り入れます。以上が、要件定義の一般的な手法の概要です。プロジェクトの性質や規模によって、要件定義手法は変化する場合があります。	0	0.72902	0.80778	90	85	両方の文章は要件定義のプロセスを説明しており、プロジェクトの成功に不可欠なステップであることを強調しています。ステークホルダーの特定、要件の収集と分析、文書化、確認と承認といった主要な要素が共通しており、全体的な意味合いは非常に類似しています。ただし、入力文2は要件定義の手法がプロジェクトの性質や規模によって変化する可能性があることを示唆しており、若干の追加情報があります。	両方の文は、要件定義のプロセスに関する手順を説明しており、ステークホルダーの特定、要件の収集、要件の文書化、要件の確認と承認といった共通のステップを含んでいます。入力文1はプロジェクトの背景と目的の明確化を含む一方で、入力文2は情報収集の手法や要件の分類について詳しく述べています。全体として、プロセスの流れや目的は非常に類似していますが、細部の焦点や説明の深さに若干の違いがあります。
36	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区別する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 1の要件定義パターン要件定義には、上記で紹介した一般的な手法以外にも、以下のような異なるパターンが存在します。1. **アジャイル開発における要件定義** : アジャイル開発では、要件定義を継続的なプロセスとして捉えます。一度に全ての要件を定義するのではなく、スプリントごとに必要な要件を追加・調整していきます。2. **ユースケース駆動開発 (Use Case Driven Development)** : ユースケースを中心に、要件を設定・定義します。システムの利用者やアクターが実際に行う活動に重点を置いた要件定義手法です。3. **プロトタイプング** : プロトタイプを作成し、利害関係者や利用者と共に試験・フィードバックを受けながら、要件を洗練させていく手法です。4. **ワークショップ** : 開発チームやステークホルダーとのワークショップを通じて、要件を共同で収集・定義します。チーム全体が参加することで、コミュニケーションを促進し共通理解を深めます。5. **顧客インタビュー** : 顧客や利用者との直接的なインタビューを通じて、要件を明確にし、優先順位付けを行います。リアルなフィードバックを取り入れながら要件を洗練させていきます。これらは、要件定義を柔軟に行うためのいくつかの代替パターンです。プロジェクトの目標や状況に応じて、最適な要件定義方法を選択することが重要です。	0	0.64687	0.58195	60	45	両方の文章は要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は特定の要件定義プロセスを詳細に説明しており、プロジェクトの背景から要件の確認と承認までの流れを示しています。一方、入力文2は異なる要件定義の手法を紹介し、アジャイル開発やプロトタイプングなどの柔軟なアプローチを説明しています。したがって、要件定義というテーマは共通していますが、具体的な内容や焦点が異なるため、意味合い的な類似度は中程度と評価しました。	入力文1と入力文2は、どちらも要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は、伝統的な要件定義の手順を詳細に説明しており、プロジェクトの背景から要件の確認と承認までの一連のプロセスを順序立てて示しています。一方、入力文2は、アジャイル開発やユースケース駆動開発など、異なる要件定義のパターンを紹介しています。両者は要件定義という共通のテーマを持っていますが、具体的な手法やアプローチが異なるため、意味合い的な類似度は中程度と評価しました。
37	—	—	システム開発の振り返りを効果的に行うためには以下の手順が役立ちます: ### 1. **目標の確認** - 開発プロジェクトの目標が達成されたかどうかを確認します。### 2. **プロジェクトの成功要因と課題の洗い出し** - 成功要因と現れた課題を特定し、分析します。### 3. **フィードバックの収集** - 開発チームや関係者からフィードバックを収集し、意見を整理します。### 4. **プロセスの評価** - 開発プロセスと運用を評価し、改善点を特定します。### 5. **成果物の評価** - 開発されたソフトウェアや成果物の品質と適合性を確認します。### 6. **振り返り会議の開催** - チーム全体で振り返り会議を開催し、経験を共有し、次のプロジェクトに活かせる教訓を引き出します。### 7. **改善計画の策定** - 顕著な問題点を解決するための具体的な改善計画を立てます。### 8. **成果の文書化** - 振り返り結果や改善計画を文書化し、関係者と共有します。これらこのステップを踏むことで、今後のプロジェクトの成功につながる洞察を得ることができます。	### システム開発の振り返り方法システム開発のプロジェクトを振り返る際には、次のステップを考慮すると良いでしょう: 1. **目標の確認** : まず、プロジェクトの始まりに設定された目標や要件を振り返りましょう。2. **成果物のレビュー** : 開発したシステムやソフトウェアを詳細に検討し、機能や品質について評価します。3. **チームのフィードバック** : プロジェクトに携わった全ての人からのフィードバックを収集し、プロセスやチームのパフォーマンスについて理解します。4. **課題の特定** : プロジェクト中に生じた問題や課題を明らかにし、今後の改善点として議論します。5. **成功要因の明確化** : プロジェクトの成功した部分や良かった点を特定し、将来のプロジェクトに活かすために考察します。6. **次のステップの計画** : 振り返りの結果を踏まえて、次のプロジェクトやフェーズに移る際のアクションプランを立てます。以上のステップを通じて、システム開発プロジェクトの振り返りを効果的に行うことができます。	0	0.88022	0.88923	90	85	両方の文章は、システム開発プロジェクトの振り返りを効果的に行うための方法を説明しています。目標の確認、フィードバックの収集、課題の特定、成功要因の明確化など、主要なテーマが共通しています。細部の表現や順序に違いはあるものの、伝えたい内容や目的はほぼ3です。	両方の文は、システム開発プロジェクトの振り返りに関する手順を説明しており、基本的なステップや目的が非常に類似しています。具体的には、目標の確認、成果物の評価、フィードバックの収集、課題の特定、成功要因の明確化、次のステップの計画といった要素が共通しています。ただし、入力文1はより詳細な手順を提供し、振り返り会議の開催や成果の文書化といった追加のステップが含まれているため、完全に一致しているわけではありません。これらの違いがスコアに反映されています。

付録2：実験データと評価結果 (一部)

- ・NoはデータのユニークIDです。
- ・文脈的に合致した同じ文章をtext1とtext2の両方に挿入し、文章の長文化をしたもののセットを長文化セット列で表し値が一致するものがセットです。長文化種別は、元の短文か、長文化後の文章かを表しています。
- ・text1, text2は類似度を測る対象の文章です。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1, 別=0で表しています。
- ・text-embedding-3-smallから、ChatGPTスコアは各モデルでの類似度評価結果を表します。text-embedding系モデルの2種は0~1で表し、1に近いほど類似度が高いです。ChatGPT系モデルの2種は0~100で表し、100に近いほど類似度が高いです。
- ・ChatGPT系モデルは評価値の根拠も同時に出力したため、掲載しています。

No	長文化種別	長文化セット	text1	text2	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	ChatGPT比較類似度の根拠	ChatGPT独立類似度の根拠
1	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイル開発は反復的で柔軟性があり、変化に迅速に対応できるのに対し、ウォーターフォール開発は計画に基づいた直線的なプロセスで、各工程を順に完了させてから次に進む点が大きな違いです。	1	0.97910	0.98436	95	95	両方の文はアジャイル開発とウォーターフォール開発の違いを説明しており、意味合いはほぼ3です。どちらもアジャイル開発の柔軟性とウォーターフォール開発の計画重視を強調しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。	両方の文は、アジャイル開発とウォーターフォール開発の特徴と違いを説明しています。具体的には、アジャイル開発の反復性と柔軟性、変化への適応力を強調し、ウォーターフォール開発の計画重視と直線的なプロセスを指摘しています。表現の違いはあるものの、伝えている内容は非常に類似しています。
2	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	アジャイルは反復的で柔軟性が高く、変更対応を重視します。一方、ウォーターフォールは各工程を順序立てて進め、後戻りが難しい計画重視の手法です。	1	0.91004	0.89423	95	85	両方の文はアジャイル開発とウォーターフォール開発の特徴を説明し、それぞれの開発手法の違いを強調しています。アジャイルの「反復的」「柔軟性」「変更対応」とウォーターフォールの「計画重視」「順序立て」「後戻りが難しい」という点が共通しており、意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の特徴を比較しており、アジャイルの反復性と柔軟性、ウォーターフォールの計画重視と直線的な進行について言及しています。違いを強調する点で内容が非常に類似していますが、表現の仕方や細部に若干の違いがあります。
3	—	—	アジャイル開発は反復的で柔軟性が高く、変更に適応しやすいのに対し、ウォーターフォール開発は直線的で計画重視、各工程が順次完了するまで次に進めない点が大きな違いです。	ウォーターフォールモデルは工程が順序立てられているため、柔軟性に欠け、要件変更や市場変化への対応が困難でした。この課題を解決するため、**反復的かつ適応的な開発**を重視するアジャイル開発が誕生しました。アジャイルは顧客との連携や短期間の反復（スプリント）を通じて、変化に対応しながら継続的に価値を提供する手法として進化しました。	0	0.76930	0.73989	85	85	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てており、特に柔軟性や適応性の違いを強調しています。文2はアジャイル開発の誕生背景を詳しく述べているため、文1よりも詳細ですが、基本的な意味合いは非常に類似しています。	両方の文はアジャイル開発とウォーターフォール開発の違いに焦点を当てています。特に、アジャイル開発の柔軟性と反復性、ウォーターフォール開発の計画重視と柔軟性の欠如について述べています。両者ともにアジャイル開発がウォーターフォール開発の課題を解決するために生まれたことを示唆しており、基本的な意味合いは非常に類似しています。ただし、文2はアジャイル開発の進化や顧客との連携についても詳しく述べているため、若干の情報の違いがあります。
4	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：ユーザーは非対称暗号方式を用いて「公開鍵」と「秘密鍵」のペアを生成します。秘密鍵は安全に保管し、公開鍵は認証サーバや接続先に共有します。2. **公開鍵の登録**：ユーザーは公開鍵を接続先サーバに登録します。登録された公開鍵は、サーバ上の認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証の開始**：ユーザーがサーバに接続を試みると、サーバは認証の一環としてランダムなデータ（チャレンジ）をユーザーに送信します。4. **秘密鍵による署名**：ユーザーは受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵を用いて署名を検証します。このプロセスで、ユーザーが秘密鍵の正当な所有者であることが確認されます。6. **接続の許可**：認証が成功すると、サーバは接続を許可し、安全な通信が確立されます。この手順により、パスワードを使わずに高いセキュリティで認証が行われます。	1	0.97842	0.96964	98	95	両方の文章は、公開鍵認証の手順を説明しており、伝えたい内容や意味はほぼ同一です。細かい表現の違いはあるものの、全体としての意味合いに大きな差異はありません。	両方の文は公開鍵認証の手順について詳細に説明しており、手順の流れや内容が非常に類似しています。鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可といったステップが3順序で記載されています。違いは、用語の選択や表現のわずかな違いに過ぎず、全体の意味合いにはほとんど影響を与えません。したがって、意味合い的な類似度は非常に高いと評価できます。
5	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. ユーザーが鍵ペア（公開鍵と秘密鍵）を生成します。 2. 公開鍵をサーバーに登録します。 3. ユーザーが接続を試みると、サーバーは暗号化されたチャレンジを送信します。 4. ユーザーは秘密鍵でチャレンジを復号し、応答を返します。 5. サーバーが応答を確認し、認証を完了します。	0	0.89827	0.87419	90	85	両方の文章は公開鍵認証の基本的な手順を説明しており、意味合いとしては非常に類似しています。どちらも鍵ペアの生成、公開鍵の登録、チャレンジと応答のプロセスを通じて認証が行われることを述べています。違いは、入力文1がより詳細で技術的な説明を含んでいる点です。	両方の文は公開鍵認証の手順について説明しており、基本的な流れは非常に似ています。鍵ペアの生成、公開鍵の登録、チャレンジの送信と応答、そして認証の完了という主要なステップが共通しています。ただし、入力文1は手順をより詳細に説明しており、特に「署名」と「署名の検証」という用語を使用しているのに対し、入力文2は「復号」と「応答を返す」という表現を用いています。この違いが若干の意味合いの違いを生んでいますが、全体としては3プロセスを指しているため、高い類似度があります。
6	—	—	公開鍵認証の手順は以下の通りです：1. **鍵ペアの生成**：利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録**：利用者は公開鍵を接続先サーバ（認証サーバなど）に登録します。公開鍵はサーバの認証ファイル（例：`~/ssh/authorized_keys`）に保存されます。3. **認証プロセスの開始**：利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ（チャレンジ）を送信します。4. **秘密鍵で署名**：利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証**：サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可**：認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです：1. サーバーがユーザーに暗号化されたチャレンジを送信します。 2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバーに渡そうとします（実際は公開鍵を登録します）。 3. サーバーは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。 4. ユーザーが秘密鍵を使用して復号し、登録プロセスをやり直します。 5. 最終的に公開鍵がサーバーに登録され、接続が試みられます。	0	0.88345	0.83765	60	45	両方の文章は公開鍵認証のプロセスを説明していますが、具体的な手順や用語の使い方に違いがあります。入力文1は正確な手順を説明しているのに対し、入力文2は一部の手順が誤解を招く表現になっており、特に秘密鍵の取り扱いに関する記述が異なります。これにより、全体的な意味合いに違いが生じています。	両方の文は公開鍵認証の手順について説明していますが、手順の詳細や順序に違いがあります。入力文1は、鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可という具体的なステップを順序立てて説明しています。一方、入力文2は手順が曖昧で、特に秘密鍵の取り扱いに誤解を招く表現があります（秘密鍵をサーバに渡そうとするなど）。また、手順の順序も異なり、鍵ペアの生成が認証プロセスの途中で促されるなど、実際の公開鍵認証の流れとは異なっています。したがって、意味合い的な類似度は中程度と評価しました。
7	—	—	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なるアプローチを採用しており、柔軟なスキーマ設計と高いスケラビリティを特徴とする非リレーショナルなデータベースです。	1	0.90452	0.89878	95	95	両方の文はNoSQLの特徴を説明しており、非リレーショナルデータベースであること、柔軟なスキーマ設計、スケラビリティの高さを強調しています。意味合いはほぼ3ですが、入力文2はリレーショナルデータベースとの対比を明示している点で若干の違いがあります。	両方の文は、NoSQLが非リレーショナルデータベースであること、柔軟なスキーマ設計を持つこと、高いスケラビリティを特徴とすることを述べています。文2は、リレーショナルデータベースとの違いを明示していますが、全体的な意味合いは非常に類似しています。

8	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	NoSQLは、リレーショナルデータベースとは異なり、柔軟なデータモデルを採用したデータベースの総称です。スケラビリティや高速性に優れ、文書型、キー・値型、カラム型、グラフ型などの種類があります。	0	0.83348	0.82690	85	85	両方の文はNoSQLの特徴として「柔軟性」と「スケラビリティ」を挙げており、NoSQLがリレーショナルデータベースとは異なることを強調しています。入力文2はさらに「高速性」や具体的なデータモデルの種類についても触れていますが、全体的な意味合いは非常に類似しています。	両方の文はNoSQLデータベースについて述べており、非リレーショナルであること、柔軟なデータモデルやスキーマ設計を持つこと、スケラビリティに優れていることを共通して説明しています。入力文2はさらにNoSQLの種類や高速性についても言及していますが、基本的な特徴に関する情報は一致しているため、高い類似度があります。
9	-	-	NoSQLは、非リレーショナルデータベースで、柔軟なスキーマ設計やスケラビリティに優れています。	SQLに「No」を突き付ける人々は、スケラビリティや柔軟性を重視し、NoSQL(非リレーショナルデータベース)を支持する傾向があります。彼らは、大量データ処理や動的スキーマ、水平分散が必要なユースケース(例:リアルタイム分析や分散型アプリ)でSQLの制約を問題視します。MongoDBやCassandraなどのNoSQLは、特定の用途でSQLを補完・代替する選択肢として注目されています。	0	0.71857	0.74304	75	85	両方の文はNoSQLの特徴である「スケラビリティ」や「柔軟性」に言及しており、NoSQLが非リレーショナルデータベースであることを共通して述べています。ただし、入力文2はNoSQLを支持する理由や具体的なユースケースについても詳しく述べているため、意味合い的にはより広範な内容を含んでいますが、基本的なテーマは一致しています。	両方の文はNoSQLデータベースに関する内容であり、特にスケラビリティや柔軟なスキーマ設計の利点を強調しています。入力文1はNoSQLの特徴を簡潔に述べており、入力文2はNoSQLを支持する理由や具体的なユースケースを詳しく説明しています。内容の焦点が一致しており、NoSQLの利点に関する理解が共通しているため、高い類似度があります。ただし、入力文2はSQLとの比較や具体例を含むため、詳細度に差があります。
10	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	要件定義や設計段階での問題が原因である可能性が高いです。十分に練られていない要件定義や設計上のミスが、テスト工程における障害の多発に直結します。	1	0.92471	0.91897	95	95	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であるという3意味を伝えています。表現の違いはあるものの、伝えたい内容はほぼ同一です。	両方の文は、要件定義や設計段階での問題がテスト工程での障害多発の原因であることを述べています。言い回しや表現に若干の違いはありますが、伝えたい内容や論点は非常に類似しています。
11	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	ウォーターフォール開発でシステムテスト工程で障害が多発する場合、主に**要件定義**や**設計**に問題がある可能性が高いです。要件の不備や曖昧さが原因で正しい仕様が実装されていない場合や、設計段階でのミスがテスト工程で顕在化します。また、**結合テスト**での検証不足や、単体テストでの抜け漏れも要因となり得ます。	0	0.75388	0.80472	85	85	両方の文は、要件定義や設計の不備がテスト工程での障害多発の原因であることを示しています。入力文2はより具体的な開発プロセス(ウォーターフォール開発)やテスト工程の詳細を含んでいますが、基本的な意味合いは一致しています。	両方の文は、要件定義や設計工程に問題があるとテストで障害が多発する可能性が高いことを述べています。入力文1は要件定義や設計ミスが直接テストでの障害につながると述べており、入力文2はウォーターフォール開発におけるシステムテストでの障害多発の原因として要件定義や設計の問題を挙げています。入力文2はさらに結合テストや単体テストの不足も要因として挙げており、詳細な説明が加えられていますが、基本的な意味合いは非常に類似しています。
12	-	-	要件定義や設計工程に問題がある可能性が高いです。不十分な要件定義や設計ミスがテストでの障害多発につながります。	システムで障害が多発するケースには、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障が挙げられます。特にスケラビリティを考慮しない設計や負荷テスト不足は、アクセス急増時に問題を引き起こします。また、運用プロセスの不備や変更管理の失敗、冗長性の欠如も影響します。継続的な監視や定期的なメンテナンスが重要です。	0	0.65997	0.53354	60	60	両方の文はシステムやプロジェクトにおける問題の原因について述べていますが、入力文1は要件定義や設計ミスに特化しており、入力文2はより広範な原因をカバーしています。共通点として設計の不備が挙げられますが、入力文2は他の要因も含むため、意味合い的な類似度は中程度です。	両方の文はシステムにおける障害の原因について述べていますが、焦点が異なります。入力文1は主に要件定義や設計工程の問題に焦点を当てており、これがテストでの障害多発につながるとしています。一方、入力文2はシステム障害の原因をより広範に捉え、設計不備、負荷増大、運用ミス、ソフトウェアバグ、ハードウェア故障などを挙げています。特にスケラビリティや運用プロセスの不備についても言及しています。共通点として設計の問題が挙げられますが、入力文2はより多くの要因を考慮しており、具体的な対策についても触れています。したがって、類似度は中程度と評価しました。
13	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社で物理サーバーを所有し運用する形式であり、一方クラウドはインターネットを通じて必要なリソースを利用する形態です。	1	0.95975	0.95273	95	95	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、内容的に非常に近いです。どちらもオンプレミスが自社でのサーバー管理を指し、クラウドがインターネット経由でのリソース利用を指しているため、伝えたい内容はほぼ同一です。表現の違いはあるものの、意味の違いはほとんどありません。	両方の文は、オンプレミスとクラウドの基本的な違いを説明しており、内容的に非常に類似しています。どちらもオンプレミスが自社で物理サーバーを所有・管理する形式であること、クラウドがインターネット経由でリソースを利用する形式であることを述べています。言い回しに若干の違いはありますが、意味合いはほぼ3です。
14	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	オンプレミスは自社でサーバーやインフラを構築・管理する方式で、初期投資や保守が必要です。一方、クラウドはインターネット経由で外部サービスを利用する方式で、スケラビリティやコスト効率に優れます。	0	0.82750	0.76366	85	85	両方の文はオンプレミスとクラウドの違いを説明しており、基本的な意味合いは非常に近いです。オンプレミスは自社での管理が必要であること、クラウドはインターネット経由で利用することを述べています。入力文2は追加でスケラビリティやコスト効率についても言及していますが、全体的な意味合いは一致しています。	両方の文は、オンプレミスとクラウドの違いについて説明しています。オンプレミスについては、自社でサーバーを保有・管理する点が共通しており、クラウドについてはインターネット経由でリソースを利用する点が一致しています。違いとしては、入力文2が初期投資や保守、スケラビリティ、コスト効率についても言及しているため、詳細度に差がありますが、基本的な意味合いは非常に類似しています。
15	-	-	オンプレミスは自社で物理的にサーバーを保有・管理し、クラウドはインターネット経由でリソースを利用する形態です。	クラウド移行は、スケラビリティ、コスト効率、災害対策の強化、運用負荷の軽減を実現します。リソースを柔軟に拡張でき、初期投資を抑えつつ、セキュリティや最新技術の恩恵を享受可能です。競争力向上に寄与します。	0	0.44651	0.34687	40	30	両方の文はクラウドに関連していますが、入力文1はオンプレミスとクラウドの基本的な違いを説明しているのに対し、入力文2はクラウド移行の具体的な利点に焦点を当てています。意味的には関連性がありますが、焦点が異なるため、意味合い的な類似度は中程度です。	入力文1はオンプレミスとクラウドの基本的な違いを説明しており、主に技術的な形態に焦点を当てています。一方、入力文2はクラウド移行の利点について述べており、スケラビリティやコスト効率、競争力向上などの具体的なメリットに焦点を当てています。両者はクラウドに関連する内容を含んでいるものの、焦点が異なるため、意味合い的な類似度は低いです。
16	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデータを体系的に管理するシステム全体を指し、複数のテーブルを含む容器のようなものです。一方、テーブルはデータベース内でデータを行と列で構造化して格納する単位です。	1	0.89072	0.87697	95	95	両方の文はデータベースとテーブルの関係を説明しており、意味合いは非常に似ています。どちらもデータベースをデータを管理するシステム全体として説明し、テーブルをデータを構造化して保存する単位として説明しています。入力文2の方が若干詳細ですが、基本的な意味合いは一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であること、テーブルがデータを行と列で構造化して保存する単位であることを述べています。文2は、データベースを「複数のテーブルを含む容器」と表現しており、文1よりも少し詳細ですが、基本的な意味合いは非常に類似しています。
17	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースは、データを体系的に管理・保存するための**全体の容器**や構造を指します。一方、テーブルはデータベース内でデータを格納するための**個々の構造化された単位**です。データベースは複数のテーブルを持つことができ、それぞれのテーブルが行と列を持ち、特定の種類のデータを扱います。データベースはシステム全体、テーブルはその中の部品と考えられます。	0	0.86902	0.82861	85	95	両方の文はデータベースとテーブルの役割と関係性を説明しており、基本的な意味合いは一致しています。入力文2はより詳細で具体的な説明を提供していますが、全体的な意味は入力文1と一致しています。	両方の文は、データベースとテーブルの関係について説明しており、データベースがデータを管理するシステム全体であること、テーブルがその中でデータを構造化して保存する単位であることを述べています。入力文2は、入力文1の内容をより詳細に説明しており、データベースが複数のテーブルを持つことや、テーブルが行と列を持つことについても触れています。全体として、両方の文は3概念を伝えており、意味合い的に非常に類似しています。

18	-	-	データベースはデータを管理するシステム全体を指し、テーブルはその中でデータを行と列で構造化して保存する単位です。	データベースはデジタル情報を整理・管理するシステムで、データを効率的に保存・検索・操作できます。一方、テーブルは物理的な家具で、物を置く・作業する場を提供します。データベースは抽象的で機能的、テーブルは具体的に物理的な用途に特化しています。	0	0.79866	0.67894	40	40	両方の文は「データベース」と「テーブル」に関する情報を提供していますが、文1はデータベース内のテーブルの役割を説明しているのに対し、文2はデータベースとテーブルを異なる文脈で比較しています。文2では、データベースは情報管理システムとしての機能を強調し、テーブルは物理的な家具としての役割を説明しており、意味合い的には異なる内容を持っています。	両方の文は「データベース」と「テーブル」という用語を扱っていますが、それぞれの文での「テーブル」の意味が異なります。入力文1では、データベース内の構造化されたデータの保存単位としての「テーブル」を説明していますが、入力文2では物理的な家具としての「テーブル」を説明しています。データベースに関する説明は類似していますが、テーブルに関する説明が異なるため、全体としての意味合い的類似度は低くなります。
19	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合するテーブルの両方で一致する行のみを取得します。一方、**OUTER JOIN**は一致する行に加えて、一方のテーブルにしか存在しない行も含めることができます。	1	0.85767	0.87835	95	95	両方の文はINNERJOINとOUTERJOINの機能の違いを説明しており、基本的な意味合いは非常に近いです。INNERJOINが両テーブルで一致するデータを取得し、OUTERJOINが片方のテーブルに一致しないデータも含めるという点で、意味の一致が見られます。表現の違いはあるものの、伝えたい内容はほぼ3です。	両方の文は、データベースにおけるINNER JOINとOUTER JOINの機能を説明しています。内容としては、INNER JOINが両方のテーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルにしか存在しないデータも含めることを述べています。言い回しや細かい表現の違いはあるものの、伝えている情報はほぼ3であるため、高い類似度スコアを付けました。
20	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	**INNER JOIN**は、結合条件を満たす両方のテーブルの共通部分のみを取得します。一致しないデータは結果に含まれません。 **OUTER JOIN**は、条件を満たさないデータも含める結合で、以下の種類があります： - **LEFT OUTER JOIN**：左側のテーブルの全データと一致する右側のデータ。 - **RIGHT OUTER JOIN**：右側のテーブルの全データと一致する左側のデータ。 - **FULL OUTER JOIN**：両方のテーブルの全データを取得します。	0	0.81617	0.80728	85	85	両方の文はINNERJOINとOUTERJOINの基本的な概念を説明しており、意味合いは非常に似ています。ただし、入力文2はOUTERJOINの種類についての追加情報を提供しているため、完全に同一の意味合いではありません。	両方の文は、INNER JOINとOUTER JOINの基本的な概念を説明しており、特にINNER JOINについては非常に類似した内容を述べています。OUTER JOINに関しては、入力文2がLEFT、RIGHT、FULL OUTER JOINの詳細を追加しているため、情報量が多く、やや異なる視点を提供していますが、基本的な意味合いは一致しています。したがって、全体的な意味合い的類似度は高いと評価できます。
21	-	-	INNER JOINは両テーブルで一致するデータのみを取得し、OUTER JOINは片方のテーブルに一致しないデータも含めて取得します。	INNER JOINをOUTER JOINに書き換えるには、結合タイプを明示的に指定します。INNER JOINでは共通データのみ取得しますが、OUTER JOINでは片方のテーブルの全行を保持します。	0	0.82297	0.72906	80	85	両方の文はINNERJOINとOUTERJOINの違いについて説明していますが、入力文2は書き換え方法についても触れており、意味合いに若干の追加情報があります。それでも、基本的な意味合いは非常に似ています。	両方の文はINNER JOINとOUTER JOINの違いについて説明しています。入力文1は、INNER JOINが両テーブルで一致するデータのみを取得し、OUTER JOINが片方のテーブルに一致しないデータも含めることを述べています。入力文2も同様に、INNER JOINが共通データのみを取得し、OUTER JOINが片方のテーブルの全行を保持することを説明しています。ただし、入力文2はINNER JOINをOUTER JOINに書き換える方法についても触れており、結合タイプの指定についての情報が追加されています。このため、内容の焦点が若干異なる部分があるものの、基本的な意味合いは非常に類似しています。
22	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	用途によりますが、クラウドは初期コストが抑えられ柔軟性が高い一方で、長期的には利用状況に応じてコストが増加する可能性があります。	1	0.88793	0.93630	95	95	両方の文はクラウドの利点と欠点を述べており、初期コストの低さと柔軟性の高さを利点として挙げ、長期的なコスト増加の可能性を欠点として指摘しています。意味合いはほぼ3ですが、文2では「可能性があります」と若干の不確実性を示しています。	両方の文は、クラウドの特性について述べており、初期コストの低さと柔軟性の高さ、そして長期的なコスト増加の可能性について言及しています。表現の違いはあるものの、伝えたい内容は非常に類似しています。
23	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	長期的なコスト効率は利用状況によりますが、一般的に次のように考えられます：- **クラウド**：初期投資が少なく、使用量に応じた課金モデルのため、柔軟性が高くスケラブルな運用が可能。ただし、長期で大規模な利用ではランニングコストが増える場合も。- **オンプレミス**：初期投資が大きい一方、長期間にわたって一定規模の利用を続ける場合、運用コストが安定し、総コストが抑えられることがあります。ただし、保守・更新コストが追加される点に留意。必要な規模や用途によってどちらが効率的かが異なります。	0	0.78180	0.72671	75	85	両方の文はクラウドの初期コストの低さと長期的なコスト増加の可能性について述べています。意味合いとしては、クラウドのコストに関する利点と欠点を説明している点で類似していますが、入力文2はオンプレミスとの比較も含んでおり、より広範な視点を提供しています。	両方の文はクラウドのコストと柔軟性に関する内容を扱っており、特に初期コストの低さと利用量に応じたコスト増加について言及しています。入力文2はオンプレミスについても触れていますが、クラウドに関する部分は入力文1と非常に類似しています。したがって、意味合い的な類似度は高いと評価できます。
24	-	-	用途次第です。クラウドは初期コストが低く柔軟性が高い一方、長期的には利用量に応じたコストが増加します。	オンプレミスは、一度の設備投資で長期間安定稼働が可能で、クラウドのような毎月のランニングコストが発生しません。大容量データや高負荷のアプリケーションでも、自社内で管理することで追加コストを抑えられます。特に既存設備の活用やカスタマイズ性が求められる環境では、オンプレミスが最も合理的で経済的な選択です！	0	0.56646	0.58247	50	65	両方の文章はITインフラの選択肢に関する情報を提供していますが、焦点が異なります。入力文1はクラウドの特性に焦点を当てており、入力文2はオンプレミスの利点を強調しています。どちらもコストと柔軟性に関する情報を含んでいますが、比較の対象が異なるため、意味合い的には部分的にしか重なりません。	両方の文はITインフラの選択肢であるクラウドとオンプレミスの特徴を説明しています。入力文1はクラウドの利点とコストの増加について述べており、入力文2はオンプレミスの利点とコスト管理について述べています。どちらもコストと柔軟性に関する情報を提供しており、ITインフラの選択に関する議論の一部を形成しています。ただし、具体的な内容や視点が異なるため、完全に一致するわけではありません。
25	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は、工程を順番に進める直線的な開発モデルです。主な流れは以下の通りです： 1. **要件定義**：システムの要件や仕様を明確に定義します。 2. **設計**：システム全体の構造を設計し、詳細な設計も行います。 3. **実装**：設計内容に基づいてプログラムを開発します。 4. **テスト**：システムが要件を満たしているかを確認します。 5. **運用と保守**：システムをリリースし、運用や問題修正を行います。 各工程が完了してから次に進む仕組みのため、途中で工程を見直す際には大きな手間がかかります。	1	0.98439	0.97103	98	95	両方の文章はウォーターフォール開発の特徴と工程を説明しており、意味合いに大きな違いはありません。用語の若干の違い(例:「要求定義」vs「要件定義」)や表現の微細な違いはありますが、伝えたい内容はほぼ3です。	両方の文はウォーターフォール開発モデルについて説明しており、工程の順序や内容がほぼ3です。用語の違い(「要求定義」vs「要件定義」など)や表現のわずかな違いはありますが、全体的な意味合いは非常に類似しています。工程の進め方や見直しの手間についても同様の説明がされています。
26	-	-	ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**：システムの要件や仕様を明確化します。 2. **設計**：システム全体の構造を設計し、詳細設計を行います。 3. **実装**：設計に基づきプログラムをコーディングします。 4. **テスト**：システムが要件を満たしているか確認します。 5. **運用・保守**：リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。	ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進みます：1. **要件定義**：システムの要件や仕様を明確化する。2. **基本設計**：システム全体の構造や外部仕様を設計する。3. **詳細設計**：プログラム単位の具体的な設計を行う。4. **実装(コーディング)**：設計に基づきプログラムを作成する。5. **単体テスト**：個1のプログラムやモジュールをテストする。6. **結合テスト**：モジュール同士を結合して動作を確認する。7. **システムテスト**：システム全体の動作を確認し、要件通りか検証する。8. **運用・保守**：完成したシステムを運用し、問題が発生した場合に修正する。各工程が順に完了しないと次に進めないのが特徴です。	0	0.93794	0.93866	90	85	両方の文章はウォーターフォール開発の基本的な考え方と工程の流れを説明しており、意味合いは非常に似ています。どちらも工程を順序立てて進めることの重要性を強調していますが、入力文2は工程をより詳細に説明しているため、若干の違いがあります。	両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要求定義から運用・保守までの流れを説明しており、全体的なプロセスの流れは非常に似ています。ただし、入力文2はより詳細な工程(基本設計、詳細設計、単体テスト、結合テスト、システムテスト)を含んでおり、工程の細分化がされています。この違いがスコアに影響していますが、全体的な意味合いは非常に類似しています。

27	-	-	<p>ウォーターフォール開発は、工程を順序立てて進める開発モデルです。基本的な流れは次の通りです。 1. **要求定義**: システムの要件や仕様を明確化します。 2. **設計**: システム全体の構造を設計し、詳細設計を行います。 3. **実装**: 設計に基づきプログラムをコーディングします。 4. **テスト**: システムが要件を満たしているか確認します。 5. **運用・保守**: リリース後のシステム運用と問題修正を行います。 各工程を完了後に次へ進むため、工程の見直しには手間がかかります。</p>	<p>ウォーターフォール開発は各工程を順序立てて進める開発手法で、以下の順序で進行します: 5. **単体テスト**: 個1のプログラムやモジュールをテストする。 1. **要件定義**: システムの要件や仕様を明確化する。 7. **システムテスト**: システム全体の動作を確認し、要件通りか検証する。 3. **詳細設計**: プログラム単位の具体的な設計を行う。 8. **運用・保守**: 完成したシステムを運用し、問題が発生した場合に修正する。 4. **実装(コーディング)**: 設計に基づきプログラムを作成する。 6. **結合テスト**: モジュール同士を結合して動作を確認する。 2. **基本設計**: システム全体の構造や外部仕様を設計する。 各工程が順に完了しないと次に進めないのが特徴です。</p>	0	0.91345	0.91544	90	85	<p>両方の文はウォーターフォール開発モデルの基本的な概念とその工程について説明しています。両方も、工程が順序立てて進行し、各工程が完了しないと次に進めないという特徴を強調しています。意味合い的には非常に類似しています。が、入力文2はテスト工程をより詳細に説明しており、若干の違いがあります。それでも、全体的な意味合いはほぼ3です。</p>	<p>両方の文はウォーターフォール開発モデルについて説明しており、工程を順序立てて進めるという基本的な特徴を共有しています。入力文1と入力文2は、要件定義、設計、実装、テスト、運用・保守といった主要な工程を含んでおり、全体的な流れも似ています。ただし、入力文2はテスト工程をより詳細に分けて説明しており、工程の順序が異なる部分もありますが、このため、完全に一致するわけではありませんが、基本的な概念と流れは非常に類似しています。</p>
28	-	-	<p>Javaプログラムの動作順序は以下の通りです: 1. **ソースコード作成**: Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**: `javac`コンパイラを使用してソースコードをコンパイルし、バイトコード(`.class`ファイル)に変換。3. **クラスローダーによる読み込み**: JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**: JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**: プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。</p>	<p>Javaプログラムの実行手順は以下の通りです: 1. **ソースコード作成**: Javaでプログラムを記述し、`java`拡張子のファイルとして保存します。 2. **コンパイル**: `javac`コンパイラを使ってソースコードをバイトコード(`.class`ファイル)に変換します。 3. **クラスローダーによるロード**: JVM (Java仮想マシン) がバイトコードを読み込み、メモリ上に展開します。 4. **バイトコードの実行**: JVM内でインタプリタまたはJIT (Just-In-Time) コンパイラがバイトコードをネイティブコードに変換し、実行します。 5. **結果の表示**: プログラムの実行結果が出力されます。 このプロセスにより、Javaはプラットフォームに依存せず動作する特性を持っています。</p>	1	0.90186	0.95154	98	95	<p>両方の文は、Javaプログラムの実行手順を説明しており、内容的にはほぼ同一です。細かい表現の違い(例: "動作順序" vs "実行手順", "結果表示" vs "結果の表示")はありますが、全体の意味合いに大きな違いはありません。</p>	<p>両方の文は、Javaプログラムの実行手順について詳細に説明しており、内容的にはほぼ同一です。手順の順序や内容も一致しており、用語の違い(例えば「動作順序」と「実行手順」)や表現のわずかな違い(例えば「結果表示」と「結果の表示」)があるものの、意味合いに大きな違いはありません。したがって、意味合い的な類似度は非常に高いと評価できます。</p>
29	-	-	<p>Javaプログラムの動作順序は以下の通りです: 1. **ソースコード作成**: Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**: `javac`コンパイラを使用してソースコードをコンパイルし、バイトコード(`.class`ファイル)に変換。3. **クラスローダーによる読み込み**: JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**: JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**: プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。</p>	<p>Javaプログラムが動作する基本的な順序は以下の通りです: 1. **ソースコード作成**: Javaファイル(`.java`)にソースコードを記述します。2. **コンパイル**: `javac`コンパイラを使用して、ソースコードをバイトコード(`.class`ファイル)に変換します。 例: `javac MyProgram.java` 3. **クラスローダーによるロード**: Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。4. **バイトコードの検証**: JVMのバイトコードベリファイアがコードの安全性を確認します。5. **実行準備**: JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。6. **実行(メインメソッドの呼び出し)**: 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例: `java MyProgram` 7. **コード実行**: JVMがバイトコードを解釈(またはJITコンパイル)し、各命令をCPUで実行します。8. **終了**: プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。</p>	0	0.92306	0.92074	90	85	<p>両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。どちらもJavaのプラットフォーム非依存性を強調し、ソースコードの作成から実行までの流れを説明しています。入力文2はより詳細な説明を含んでいますが、全体的な意味合いは一致しています。</p>	<p>両方の入力文は、Javaプログラムの動作順序について説明しており、基本的な流れは非常に類似しています。具体的には、ソースコードの作成、コンパイル、クラスローダーによるロード、バイトコードの実行といった主要なステップが共通しています。ただし、入力文2にはバイトコードの検証や実行準備、プログラム終了時のメモリ解放といった詳細なステップが追加されており、より詳細なプロセスが記述されています。このため、全体的な流れは似ているものの、細部において差異があるため、完全に同一の意味合いとは言えません。</p>
30	-	-	<p>Javaプログラムの動作順序は以下の通りです: 1. **ソースコード作成**: Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**: `javac`コンパイラを使用してソースコードをコンパイルし、バイトコード(`.class`ファイル)に変換。3. **クラスローダーによる読み込み**: JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**: JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**: プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。</p>	<p>Javaプログラムが動作する基本的な順序は以下の通りです: 1. **実行準備**: JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。 2. **ソースコード作成**: Javaファイル(`.java`)にソースコードを記述します。 3. **終了**: プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。 4. **クラスローダーによるロード**: Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。 5. **コンパイル**: `javac`コンパイラを使用して、ソースコードをバイトコード(`.class`ファイル)に変換します。 例: `javac MyProgram.java` 6. **バイトコードの検証**: JVMのバイトコードベリファイアがコードの安全性を確認します。 7. **コード実行**: JVMがバイトコードを解釈(またはJITコンパイル)し、各命令をCPUで実行します。 8. **実行(メインメソッドの呼び出し)**: 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例: `java MyProgram` この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。</p>	0	0.89679	0.87155	80	75	<p>両方の文章はJavaプログラムの動作プロセスを説明しており、基本的な意味合いは非常に似ています。どちらもJavaがプラットフォームに依存せずに動作することを強調しています。ただし、入力文2はより詳細な手順を含んでおり、特定のプロセス(例: バイトコードの検証、メインメソッドの呼び出し)についての追加情報を提供しています。これにより、意味合い的な細部に若干の違いがありますが、全体的な意味合いは一致しています。</p>	<p>両方の文はJavaプログラムの動作順序について説明していますが、詳細なステップや順序の違いがあります。入力文1は、ソースコード作成から結果表示までの基本的な流れを簡潔に説明しています。一方、入力文2は、実行準備やバイトコードの検証、メインメソッドの呼び出しなど、より詳細なステップを含んでいます。両方の文がJavaのプラットフォーム非依存性に言及している点は共通していますが、具体的な手順の記述に違いがあるため、完全に一致するわけではありません。したがって、意味合い的な類似度は75と評価しました。</p>
31	-	-	<p>ウォーターフォール開発は以下の順序で進行します: 1. **要件定義**: システムの要件を明確化。 2. **基本設計**: 全体の構造を設計。 3. **詳細設計**: プログラム単位の具体設計。 4. **実装**: 設計に基づきプログラム作成。 5. **単体テスト**: 各プログラムをテスト。 6. **結合テスト**: モジュール間の動作確認。 7. **システムテスト**: システム全体を検証。 8. **運用・保守**: 問題修正や改善を行う。順序を厳密に守ることが特徴です。</p>	<p>ウォーターフォール開発は以下の順序で進行します: 3. **詳細設計**: プログラム単位の具体設計。 7. **システムテスト**: システム全体を検証。 1. **要件定義**: システムの要件を明確化。 6. **結合テスト**: モジュール間の動作確認。 4. **実装**: 設計に基づきプログラム作成。 8. **運用・保守**: 問題修正や改善を行う。 2. **基本設計**: 全体の構造を設計。 5. **単体テスト**: 各プログラムをテスト。 順序を厳密に守ることが特徴です。</p>	0	0.96852	0.95236	95	85	<p>両方の文はウォーターフォール開発のプロセスとその特徴である「順序を厳密に守ること」を説明しています。ほぼ3です。</p>	<p>両方の文はウォーターフォール開発のプロセスを説明しており、3ステップを含んでいます。しかし、入力文2ではステップの順序が異なっており、ウォーターフォール開発の特徴である「順序を厳密に守ること」が反映されていません。このため、意味合い的には類似していますが、順序の重要性が異なるため完全に一致しません。</p>
32	-	-	<p>Javaプログラムの動作順序は以下の通りです: 1. **ソースコード作成**: Javaでコードを記述し、`java`ファイルとして保存。2. **コンパイル**: `javac`コンパイラを使用してソースコードをコンパイルし、バイトコード(`.class`ファイル)に変換。3. **クラスローダーによる読み込み**: JVMがバイトコードをロードし、メモリに展開。4. **バイトコードの実行**: JVM内でインタプリタまたはJITコンパイラがバイトコードをネイティブコードに変換して実行。5. **結果表示**: プログラムの出力が表示される。この順序により、Javaはプラットフォームに依存せず動作します。</p>	<p>Javaプログラムが動作する基本的な順序は以下の通りです: 1. **実行準備**: JVMがプログラムの実行に必要なメモリ領域を確保し、初期化を行います。 2. **ソースコード作成**: Javaファイル(`.java`)にソースコードを記述します。 3. **終了**: プログラムが終了すると、JVMはメモリを解放し、プロセスを終了します。 4. **クラスローダーによるロード**: Java Virtual Machine (JVM) が必要な`.class`ファイルをメモリにロードします。 5. **コンパイル**: `javac`コンパイラを使用して、ソースコードをバイトコード(`.class`ファイル)に変換します。 例: `javac MyProgram.java` 6. **バイトコードの検証**: JVMのバイトコードベリファイアがコードの安全性を確認します。 7. **コード実行**: JVMがバイトコードを解釈(またはJITコンパイル)し、各命令をCPUで実行します。 8. **実行(メインメソッドの呼び出し)**: 実行のエントリポイントである`main`メソッドが呼び出され、プログラムが開始します。 例: `java MyProgram` この順序により、Javaプログラムはプラットフォームに依存しない形で動作します。</p>	0	0.89726	0.87175	80	75	<p>両方の文章はJavaプログラムの実行プロセスを説明しており、基本的な意味合いは3です。どちらもJavaのプラットフォーム非依存性を強調しています。ただし、入力文2はより詳細な手順を含んでおり、特に「実行準備」や「バイトコードの検証」など、入力文1にはない具体的なプロセスが追加されています。これにより、意味合いに若干の違いが生じていますが、全体としては3テーマを扱っています。</p>	<p>両方の文はJavaプログラムの動作順序について説明していますが、詳細なステップや順序の違いがあります。入力文1は、ソースコード作成から結果表示までの基本的な流れを簡潔に説明しています。一方、入力文2は、実行準備やバイトコードの検証、メインメソッドの呼び出しなど、より詳細なステップを含んでいます。両方の文がJavaのプラットフォーム非依存性に言及している点は共通していますが、具体的なプロセスの説明に違いがあるため、完全に一致するわけではありません。したがって、意味合い的な類似度は75と評価しました。</p>

33	—	—	公開鍵認証の手順は以下の通りです: 1. **鍵ペアの生成** : 利用者は非対称暗号方式で「公開鍵」と「秘密鍵」の鍵ペアを生成します。秘密鍵は安全に保管し、公開鍵を認証サーバや他者に共有します。2. **公開鍵の登録** : 利用者は公開鍵を接続サーバ(認証サーバなど)に登録します。公開鍵はサーバの認証ファイル(例: ~/.ssh/authorized_keys)に保存されます。3. **認証プロセスの開始** : 利用者がサーバに接続を試みると、サーバは利用者にランダムなデータ(チャレンジ)を送信します。4. **秘密鍵で署名** : 利用者は受け取ったチャレンジを秘密鍵で暗号化し、署名としてサーバに返送します。5. **署名の検証** : サーバは登録済みの公開鍵で署名を検証し、利用者が秘密鍵の正当な所有者であることを確認します。6. **接続許可** : 認証が成功すると、サーバは接続を許可します。このプロセスにより、安全かつパスワード不要で認証が行われます。	公開鍵認証の手順は以下の通りです: 1. サーバーがユーザーに暗号化されたチャレンジを送信します。 2. 公開鍵を生成した後、ユーザーは秘密鍵をサーバーに渡そうとします(実際は公開鍵を登録します)。 3. サーバーは応答を確認し、認証を完了する前に鍵ペアを生成するよう促します。 4. ユーザーが秘密鍵を使用して復号し、登録プロセスをやり直します。 5. 最終的に公開鍵がサーバーに登録され、接続が試みられます。	0	0.88341	0.83840	50	45	両方の文章は公開鍵認証のプロセスを説明しようとしていますが、入力文2には誤解を招く表現や不正確な情報が含まれています(例:秘密鍵をサーバーに渡そうとするなど)。そのため、意味合い的には一致していない部分が多く、類似度は低くなります。	両方の文は公開鍵認証の手順について説明していますが、手順の詳細や順序に違いがあります。入力文1は、鍵ペアの生成、公開鍵の登録、認証プロセスの開始、秘密鍵での署名、署名の検証、接続許可という具体的なステップを順序立てて説明しています。一方、入力文2は手順が曖昧で、秘密鍵をサーバーに渡そうとするなど、実際のプロセスとは異なる記述があります。全体として、両方の文は公開鍵認証に関連していますが、具体的な手順や正確さにおいて大きな違いがあります。
34	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区1する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 要件定義の流れ1. **プロジェクトの背景理解** - プロジェクトの目的、スコープ、ステークホルダーを理解する。2. **要件収集** - ステークホルダーとのコミュニケーションを通じて、必要な機能や要件を収集する。3. **要件分析** - 収集した要件を整理し、矛盾点や抜け漏れを特定する。4. **要件の整理と整合性の確認** - 複数の要件を整理し、矛盾や重複を解消して要求事項を洗練させる。5. **要件文書の作成** - 要求事項を記述した要件仕様書を作成し、関係者と共有する。6. **要件の検証** - 作成された要件がプロジェクトの目的やスコープと整合しているかを確認する。7. **フィードバックと調整** - ステークホルダーからのフィードバックを受け、必要に応じて要件を調整する。8. **最終的な要件の承認** - 最終的な要件がすべてのステークホルダーによって承認される。	1	0.81901	0.91259	90	85	両方の文章は、要件定義のプロセスを説明しており、プロジェクトの背景理解、要件の収集と分析、文書化、確認と承認といった主要な目的や意味合いが非常に似ています。細部の表現やステークホルダーとの関係の強調に若干の違いがありますが、全体的な意味合いはほぼ一致しています。	両方の文は要件定義のプロセスを説明しており、全体的な流れやステップが非常に類似しています。具体的には、プロジェクトの背景理解、要件の収集と分析、要件仕様書の作成、要件の確認と承認といった主要なステップが共通しています。ただし、入力文2には「フィードバックと調整」というステップが追加されており、要件の整合性確認やステークホルダーとのコミュニケーションに重点が置かれています。一方、入力文1は優先順位付けに関する詳細が含まれています。これらの違いがあるため、完全に同一の内容ではありませんが、全体的なプロセスの流れは非常に似ています。
35	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区1する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 要件定義の方法要件定義はプロジェクトの成功に不可欠なステップであり、以下の手順に従って行われます。1. **ステークホルダーの特定** : プロジェクトに関わる重要なステークホルダーを特定します。ステークホルダーには、利用者、顧客、開発チーム、マーケティング部門などが含まれます。2. **情報収集** : ステークホルダーから要件に関する情報を収集します。ヒアリング、ワークショップ、アンケートなどの手法が活用されます。3. **要件の整理と分類** : 収集した情報を整理し、共通するテーマやパターンを特定します。要件を優先順位付けして分類することが重要です。4. **要件文書の作成** : 要件を明確に文書化します。要件文書は、機能要件、非機能要件、制約要件などを含みます。5. **要件の検証** : ステークホルダーや関係者との間で要件を検証し、正確かつ完全なものであることを確認します。フィードバックを受け取り、必要に応じて修正を加えます。6. **要件の承認** : 最終的な要件文書をステークホルダーに提示し、承認を取得します。要件の変更がある場合は、適切な手続きを経て変更を取り入れます。以上が、要件定義の一般的な手法の概要です。プロジェクトの性質や規模によって、要件定義手法は変化する場合があります。	0	0.72902	0.80778	90	85	両方の文章は要件定義のプロセスを説明しており、プロジェクトの成功に不可欠なステップであることを強調しています。ステークホルダーの特定、要件の収集と分析、文書化、確認と承認といった主要な要素が共通しており、全体的な意味合いは非常に類似しています。ただし、入力文2は要件定義の手法がプロジェクトの性質や規模によって変化する可能性があることを示唆しており、若干の追加情報があります。	両方の文は、要件定義のプロセスに関する手順を説明しており、ステークホルダーの特定、要件の収集、要件の文書化、要件の確認と承認といった共通のステップを含んでいます。入力文1はプロジェクトの背景と目的の明確化を含む一方で、入力文2は情報収集の手法や要件の分類について詳しく述べています。全体として、プロセスの流れや目的は非常に類似していますが、細部の焦点や説明の深さに若干の違いがあります。
36	—	—	### 要件定義の手順1. **プロジェクトの背景と目的の明確化** - プロジェクトのコンセプトやビジョンを明確に定義する。2. **利害関係者の特定** - プロジェクトに関わるすべての利害関係者を特定し、彼らの要件や期待を理解する。3. **要件の収集** - 利害関係者から要件を収集し、優先順位を付ける。4. **要件の分析** - 収集された要件を分析し、整理して矛盾や抜け漏れがないか確認する。5. **優先順位の付与** - 要件に優先順位を与え、重要なものとそうでないものを区1する。6. **要件仕様書の作成** - 収集された要件をまとめて、要件仕様書として文書化する。7. **要件の確認と承認** - 要件仕様書を関係者に確認してもらい、必要な修正を行った後に承認を取得する。	### 1の要件定義パターン要件定義には、上記で紹介した一般的な手法以外にも、以下のような異なるパターンが存在します。1. **アジャイル開発における要件定義** : アジャイル開発では、要件定義を継続的なプロセスとして捉えます。一度に全ての要件を定義するのではなく、スプリントごとに必要な要件を追加・調整していきます。2. **ユースケース駆動開発 (Use Case Driven Development)** : ユースケースを中心に、要件を設定・定義します。システムの利用者やアクターが実際に行う活動に重点を置いた要件定義手法です。3. **プロトタイプング** : プロトタイプを作成し、利害関係者や利用者と共に試験・フィードバックを受けながら、要件を洗練させていく手法です。4. **ワークショップ** : 開発チームやステークホルダーとのワークショップを通じて、要件を共同で収集・定義します。チーム全体が参加することで、コミュニケーションを促進し共通理解を深めます。5. **顧客インタビュー** : 顧客や利用者との直接的なインタビューを通じて、要件を明確にし、優先順位付けを行います。リアルなフィードバックを取り入れながら要件を洗練させていきます。これらは、要件定義を柔軟に行うためのいくつかの代替パターンです。プロジェクトの目標や状況に応じて、最適な要件定義方法を選択することが重要です。	0	0.64687	0.58195	60	45	両方の文章は要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は特定の要件定義プロセスを詳細に説明しており、プロジェクトの背景から要件の確認と承認までの流れを示しています。一方、入力文2は異なる要件定義の手法を紹介し、アジャイル開発やプロトタイプングなどの柔軟なアプローチを説明しています。したがって、要件定義というテーマは共通していますが、具体的な内容や焦点が異なるため、意味合い的な類似度は中程度と評価しました。	入力文1と入力文2は、どちらも要件定義に関する内容を扱っていますが、アプローチが異なります。入力文1は、伝統的な要件定義の手順を詳細に説明しており、プロジェクトの背景から要件の確認と承認までの一連のプロセスを順序立てて示しています。一方、入力文2は、アジャイル開発やユースケース駆動開発など、異なる要件定義のパターンを紹介しています。両者は要件定義という共通のテーマを持っていますが、具体的な手法やアプローチが異なるため、意味合い的な類似度は中程度と評価しました。
37	—	—	システム開発の振り返りを効果的に行うためには以下の手順が役立ちます: ### 1. **目標の確認** - 開発プロジェクトの目標が達成されたかどうかを確認します。### 2. **プロジェクトの成功要因と課題の洗い出し** - 成功要因と現れた課題を特定し、分析します。### 3. **フィードバックの収集** - 開発チームや関係者からフィードバックを収集し、意見を整理します。### 4. **プロセスの評価** - 開発プロセスと運用を評価し、改善点を特定します。### 5. **成果物の評価** - 開発されたソフトウェアや成果物の品質と適合性を確認します。### 6. **振り返り会議の開催** - チーム全体で振り返り会議を開催し、経験を共有し、次のプロジェクトに活かせる教訓を引き出します。### 7. **改善計画の策定** - 顕著な問題点を解決するための具体的な改善計画を立てます。### 8. **成果の文書化** - 振り返り結果や改善計画を文書化し、関係者と共有します。これらこのステップを踏むことで、今後のプロジェクトの成功につながる洞察を得ることができます。	### システム開発の振り返り方法システム開発のプロジェクトを振り返る際には、次のステップを考慮すると良いでしょう: 1. **目標の確認** : まず、プロジェクトの始まりに設定された目標や要件を振り返りましょう。2. **成果物のレビュー** : 開発したシステムやソフトウェアを詳細に検討し、機能や品質について評価します。3. **チームのフィードバック** : プロジェクトに携わった全ての人からのフィードバックを収集し、プロセスやチームのパフォーマンスについて理解します。4. **課題の特定** : プロジェクト中に生じた問題や課題を明らかにし、今後の改善点として議論します。5. **成功要因の明確化** : プロジェクトの成功した部分や良かった点を特定し、将来のプロジェクトに活かすために考察します。6. **次のステップの計画** : 振り返りの結果を踏まえて、次のプロジェクトやフェーズに移る際のアクションプランを立てます。以上のステップを通じて、システム開発プロジェクトの振り返りを効果的に行うことができます。	0	0.88022	0.88923	90	85	両方の文章は、システム開発プロジェクトの振り返りを効果的に行うための方法を説明しています。目標の確認、フィードバックの収集、課題の特定、成功要因の明確化など、主要なテーマが共通しています。細部の表現や順序に違いはあるものの、伝えたい内容や目的はほぼ3です。	両方の文は、システム開発プロジェクトの振り返りに関する手順を説明しており、基本的なステップや目的が非常に類似しています。具体的には、目標の確認、成果物の評価、フィードバックの収集、課題の特定、成功要因の明確化、次のステップの計画といった要素が共通しています。ただし、入力文1はより詳細な手順を提供し、振り返り会議の開催や成果の文書化といった追加のステップが含まれているため、完全に一致しているわけではありません。これらの違いがスコアに反映されています。

付録3.長文化セットの評価結果

- ・短文と長文のセットで1行に記載しています。
- ・長文から短文を引いた差を同様に記載しています。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1、別=0で表しています。

短文							長文							差（長文-短文）			
NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度
66	66	1	0.95122	0.96673	95	95	132	66	1	0.99696	0.99153	98	95	0.04575	0.02479	3	0
67	67	0	0.88500	0.83577	60	70	133	67	0	0.99503	0.98875	85	95	0.11003	0.15297	25	25
68	68	1	0.84721	0.85831	90	85	134	68	1	0.99089	0.98809	98	98	0.14368	0.12978	8	13
69	69	0	0.91387	0.86809	40	60	135	69	0	0.97818	0.96897	90	95	0.06432	0.10089	50	35
70	70	0	0.93875	0.91830	70	85	136	70	0	0.99692	0.98479	90	95	0.05817	0.06648	20	10
72	72	0	0.87188	0.84732	90	75	137	72	0	0.98664	0.98259	90	95	0.11476	0.13528	0	20
73	73	0	0.82057	0.85983	90	85	138	73	0	0.95717	0.97115	95	90	0.13660	0.11131	5	5
111	111	1	0.91655	0.89746	100	95	139	111	1	0.97804	0.96826	100	95	0.06148	0.07080	0	0
112	112	1	0.96277	0.96710	95	95	140	112	1	0.97557	0.98119	98	95	0.01280	0.01409	3	0
113	113	0	0.90149	0.84192	75	70	141	113	0	0.95176	0.93341	90	85	0.05027	0.09149	15	15
114	114	1	0.85186	0.92515	95	95	142	114	1	0.98611	0.97416	100	95	0.13425	0.04901	5	0
115	115	1	0.95134	0.95962	95	95	143	115	1	0.99251	0.98502	98	95	0.04117	0.02541	3	0
116	116	0	0.94920	0.94555	70	70	144	116	0	0.99330	0.98345	90	95	0.04410	0.03789	20	25
117	117	1	0.97527	0.98402	98	95	145	117	1	0.99510	0.99397	98	98	0.01983	0.00994	0	3
118	118	1	0.95407	0.92963	95	95	146	118	1	0.98814	0.95558	98	95	0.03408	0.02595	3	0
119	119	0	0.98682	0.94501	60	85	147	119	0	0.99540	0.94425	80	95	0.00858	-0.00075	20	10
120	120	1	0.87244	0.84566	30	85	126	120	1	0.97904	0.94889	70	85	0.10660	0.10323	40	0
121	121	0	0.93091	0.87517	10	30	127	121	0	0.96875	0.93828	50	70	0.03784	0.06311	40	40
122	122	1	0.95623	0.96410	98	95	128	122	1	0.98664	0.98639	98	95	0.03041	0.02228	0	0
123	123	0	0.92170	0.88302	10	20	129	123	0	0.97480	0.90514	50	50	0.05310	0.02212	40	30
124	124	1	0.87822	0.93415	85	85	130	124	1	0.97862	0.96418	90	95	0.10040	0.03003	5	10
125	125	0	0.88555	0.86678	20	20	131	125	0	0.98770	0.97186	60	70	0.10215	0.10508	40	50

付録3.長文化セットの評価結果

- ・短文と長文のセットで1行に記載しています。
- ・長文から短文を引いた差を同様に記載しています。
- ・人間の評価は、人間が同じか、別かを判断した結果で、同じ=1、別=0で表しています。

短文							長文							差（長文-短文）			
NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	NO	長文化セット	人間の評価	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度	text-embedding-3-small	text-embedding-3-large	ChatGPT比較類似度	ChatGPT独立類似度
66	66	1	0.95122	0.96673	95	95	132	66	1	0.99696	0.99153	98	95	0.04575	0.02479	3	0
67	67	0	0.88500	0.83577	60	70	133	67	0	0.99503	0.98875	85	95	0.11003	0.15297	25	25
68	68	1	0.84721	0.85831	90	85	134	68	1	0.99089	0.98809	98	98	0.14368	0.12978	8	13
69	69	0	0.91387	0.86809	40	60	135	69	0	0.97818	0.96897	90	95	0.06432	0.10089	50	35
70	70	0	0.93875	0.91830	70	85	136	70	0	0.99692	0.98479	90	95	0.05817	0.06648	20	10
72	72	0	0.87188	0.84732	90	75	137	72	0	0.98664	0.98259	90	95	0.11476	0.13528	0	20
73	73	0	0.82057	0.85983	90	85	138	73	0	0.95717	0.97115	95	90	0.13660	0.11131	5	5
111	111	1	0.91655	0.89746	100	95	139	111	1	0.97804	0.96826	100	95	0.06148	0.07080	0	0
112	112	1	0.96277	0.96710	95	95	140	112	1	0.97557	0.98119	98	95	0.01280	0.01409	3	0
113	113	0	0.90149	0.84192	75	70	141	113	0	0.95176	0.93341	90	85	0.05027	0.09149	15	15
114	114	1	0.85186	0.92515	95	95	142	114	1	0.98611	0.97416	100	95	0.13425	0.04901	5	0
115	115	1	0.95134	0.95962	95	95	143	115	1	0.99251	0.98502	98	95	0.04117	0.02541	3	0
116	116	0	0.94920	0.94555	70	70	144	116	0	0.99330	0.98345	90	95	0.04410	0.03789	20	25
117	117	1	0.97527	0.98402	98	95	145	117	1	0.99510	0.99397	98	98	0.01983	0.00994	0	3
118	118	1	0.95407	0.92963	95	95	146	118	1	0.98814	0.95558	98	95	0.03408	0.02595	3	0
119	119	0	0.98682	0.94501	60	85	147	119	0	0.99540	0.94425	80	95	0.00858	-0.00075	20	10
120	120	1	0.87244	0.84566	30	85	126	120	1	0.97904	0.94889	70	85	0.10660	0.10323	40	0
121	121	0	0.93091	0.87517	10	30	127	121	0	0.96875	0.93828	50	70	0.03784	0.06311	40	40
122	122	1	0.95623	0.96410	98	95	128	122	1	0.98664	0.98639	98	95	0.03041	0.02228	0	0
123	123	0	0.92170	0.88302	10	20	129	123	0	0.97480	0.90514	50	50	0.05310	0.02212	40	30
124	124	1	0.87822	0.93415	85	85	130	124	1	0.97862	0.96418	90	95	0.10040	0.03003	5	10
125	125	0	0.88555	0.86678	20	20	131	125	0	0.98770	0.97186	60	70	0.10215	0.10508	40	50