生成 AI を活用した数学学習支援システムの再構築と評価

Reconstruction and Evaluation of a Mathematics Learning Support System Utilizing Generative AI

山口大学・教育学部　　北本　卓也 *1
Takuya Kitamoto
Faculty of Education, Yamaguchi University

新居高等学校　　木原　宏佳 *2
Hiroyoshi Kihara
Shizuoka Prefectural Arai Senior High School

Abstract

This study reports the refactoring of "Palette," a web-based mathematics learning system, using the AI-native IDE "Cursor" to overcome increasing architectural complexity. By utilizing Cursor's codebase indexing and "Plan Mode," we successfully integrated computer algebra systems and Gemini (LLM) to provide automated grading and interactive tutoring. Application to a differentiation drill demonstrated that this AI-integrated workflow enables the rapid development of sophisticated educational content. While AI increased efficiency for simple tasks by approximately 55%, it reduced performance for complex debugging by 19%. Consequently, human oversight remains vital for ensuring the security and accuracy of the generated code.

1　はじめに

1.1　研究の背景：数学学習支援システムの現状と課題

近年，数学のオンライン学習のためのシステム構築が重要視されている．著者らは，プログラミングの専門知識を必ずしも持たない教育者でもブラウザ上で数学学習用の Web アプリケーションを作成できる「Palette システム」の開発を行ってきた ([1],[2])．本システムは JavaScript で記述され，UI コンポーネントを自由に配置して HTML として出力できる点に特徴がある．

しかし，機能拡充に伴いシステムが複雑化し，リファクタリングによる内部構造の改善が不可欠となった．本研究では，近年急速に普及している AI コーディングツール ([3]) を活用し，システムの刷新と新機能の実装を試みた．

*1 〒 753-8513 山口市吉田 1677-1　E-mail: kitamoto@yamaguchi-u.ac.jp
*2 〒 431-0304 湖西市新居町内山 2036

1.2 開発上の課題：システム肥大化とリファクタリングの必要性

Palette システムの開発継続に伴い，多様な学習ニーズに応えるべくコンポーネントの種類を拡充してきたが，これがシステムの複雑化を招く要因となった．具体的には，機能の追加が繰り返されることでコードの内部構造が煩雑になり，保守性が低下するという問題に直面した．

この課題を解決するためには，ソフトウェアの外部から見た動作を変えることなく，内部構造を改善する「リファクタリング」が不可欠である．リファクタリングの主目的は，コードの可読性や保守性，再利用性を高めることにあり，システムの長期的な安定運用と品質維持のために重要なプロセスとなる．

1.3 生成 AI による開発パラダイムの転換

一方で，近年の生成 AI 技術の進展により，ソフトウェア開発のワークフローは急速に進展している．Stack Overflow の 2024 年の調査によれば，開発者の約 76%が AI コーディングアシスタントを利用している，または利用する予定があるとしており，AI ツールは日常的な開発において不可欠な存在となりつつある ([3])．

特に，AI がプロジェクト全体の文脈を理解し，対話を通じてコードの編集やデバッグを行う「AI ネイティブな開発環境」の登場は，リファクタリング作業の効率を大幅に向上させる可能性を秘めている．

1.4 本研究の目的

本研究の目的は，数学学習支援システム「Palette」を AI-native IDE である Cursor ([4]) を用いて再構築し，その過程で CAS および LLM を統合した新たな教材開発基盤を実現することである．あわせて，微分演習ドリルの開発事例を通して，この開発基盤が数学教材の迅速な試作と拡張にどのように寄与するかを検討する．さらに，AI を活用した開発プロセスの利点と限界を整理し，今後の教育用システム開発に向けた課題を明らかにする．

2 AI コーディングツールの現状と選定

2.1 AI ツールの普及と進化

近年，ソフトウェア開発における生成 AI の活用は急速に拡大している．Stack Overflow による 2024 年の調査では，開発者の 82.7%が AI tool が生産性を増すと回答しており，AI ツールは日常的なワークフローに不可欠な存在となっている ([3])．AI ツールの役割は，2018 年頃の単純な「コード補完（オートコンプリート）」から，2021 年の「文脈理解型アシスタント（GitHub Copilot 等）」，2023 年の「対話型ペアプログラマー（Cursor 等）」，そして 2024 年の「自律型エージェント（DEVIN 等）」へと，わずか数年で急速に進展している ([5])．

現代の AI コーディングツールは，単なるコード生成にとどまらず，デバッグ，既存コードのリファクタリング，ユニットテストの自動生成，さらには難解なコードのドキュメント作成や新技術の習得支援など，開発のあらゆるフェーズをカバーしている．GitHub の調査によれば，特に反復的なタスクにおいて AI は開発速度を大幅に引き上げる効果を発揮している．

2.2 主要ツールの比較と特性

本研究では，以下の 3 つの役割を比較検討した ：

- **GitHub Copilot**: 文脈を読み取りリアルタイムで提案を行う「アシスタント型」
- **Cursor**: プロジェクト全体を理解し対話を通じて編集を行う「AI ファースト IDE 型」
- **DEVIN**: 曖昧な指示から自律的にタスクを完了させる「エージェント型」

これらのツールを比較すると，GitHub Copilot が補完機能において高い評価を得ているのに対し，Cursor は対話能力やデバッグ支援において，DEVIN は自律性において相対的に高い傾向がある．

2.3 ツール選定：Cursor の採用

本研究における Palette システムのリファクタリングにあたっては，以下の理由から「Cursor」を選定した．

第一に,「プロジェクト全体の深い文脈理解（Project-wide Smarts）」である．Cursor はプロジェクト全体のコードをセマンティック（意味的）に分析して構造を理解するため，局所的ではない高精度な回答や編集が可能である． 第二に,「高度なエージェント機能（Plan Mode）」の存在である．高次の目的を提示することで，AI が複数ファイルにまたがる複雑なタスク（新機能追加やリファクタリング）の計画と実行を自律的に行うことができ，大規模なコード刷新に適している． 第三に,「既存資産との互換性」である．Cursor は世界的に普及している Visual Studio Code（VS Code）のフォークであり，使い慣れた UI やショートカット，既存の拡張機能をそのまま引き継ぐことができるため，導入の障壁が極めて低い．また，GPT 5 や Claude 4 といった最新の AI モデルをニーズに応じて切り替えて使用できる点も大きな利点である．

2.4 導入における戦略的リスクと課題

AI ツールの導入には多大なメリットがある一方で，重大な戦略的リスクも存在する．AI が生成したコードの約 40%に脆弱性が含まれるという報告もあり，セキュリティ負債の増大が懸念される．また，ライセンス表示なしに GPL コードを生成してしまうことによる知的財産侵害のリスク（GPL 汚染）も指摘されている．

さらに，開発者の 84%が AI ツールを利用または利用予定であるのに対し，AI の出力を積極的に信頼している層は 46%にとどまるという「導入と信頼のパラドックス」が存在する．AI は「ほぼ正しいが，完全には正しくない」コードを出力することが多く，結果としてデバッグにかえって時間がかかるという不満も根強い．本研究においても，これらの課題を背景に，AI 生成コードに対する厳格なレビュー体制を維持しつつ開発を進めた．

3 Palette システムのリファクタリングと新機能の実装

3.1 Cursor を用いたコードベースの刷新

システムの肥大化に伴う複雑性を解消するため，AI ネイティブな IDE である「Cursor」を導入し，大規模なリファクタリングを実施した．

- **プロジェクト全体の文脈認識（Codebase Indexing）**：Cursor のセマンティック検索機能を活用し，プロジェクト全体のコード構造を AI にインデックス化させた．これにより，局所的なコード修正にとどまらず，ファイル間を跨ぐ依存関係やコンポーネント間の通信プロトコルを正確に把握した状態での改修が可能となった．
- **計画主導型開発（Plan Mode）の活用**：「Plan Mode」を用いることで，実装前に「コンポーネント管理の抽象化」や「通信 API の統一」といった高次の設計方針を AI と合意した．AI が提案した複数のリファクタリング案を人間が吟味し，承認された計画に基づいて自律的にコードが生成されるプロセスを繰り返すことで，保守性の高い内部構造への刷新を実現した．

3.2 数式処理エンジン（CAS）の統合

数学学習支援システムとしての核となる計算能力を強化するため，JavaScript で動作する 2 つの数式処理システム（CAS）を統合した．

- **Algebrite の統合**: 軽量な CAS である Algebrite を採用し，ブラウザ上での因数分解や簡約化といった代数計算をリアルタイムで実行可能にした．
- **Nerdamer の統合**: より高度な数式処理が可能な Nerdamer を併載し，微積分や複雑な方程式の解法に対応させた．

これらの CAS は，サーバーサイドに計算資源を必要としないクライアントサイド実行形式をとっているため，教育現場での導入障壁が低いという利点を持つ．

3.3 コンテンツ表現の高度化（TeX と Markdown）

教材の視認性と記述効率を両立するため，以下のライブラリを用いた表現層の強化を行った．

- **KaTeX による数式レンダリング**: TeX 形式の数式を高速かつ美しく表示するため，KaTeX を採用した．これにより，動的に生成された問題文や計算結果を，即座に標準的な数学表記で出力することが可能となった．
- **Marked.js による Markdown 処理**: 解説文などのリッチテキスト作成を効率化するため，Marked.js を用いた Markdown コンポーネントを実装した．数式（KaTeX）とテキスト（Markdown）を混在させたドキュメントを容易に生成できる環境を構築した．

3.4 LLM（Gemini）の機能コンポーネント化

本システムの最大の特徴は，生成 AI を学習者への支援機能として直接統合した点にある．

- **対話型チュータリング機能**: 単なる計算結果の表示にとどまらず，誤答時やヒント要求時に LLM が問題の解法をステップバイステップで解説する機能を実装した．
- **コンポーネント間連携（PaletteComponent API）**：以下の API を通じて，LLM や CAS と他の UI 要素をシームレスに連携させた．
  - `getMainText(idname)`: 特定のコンポーネントから入力テキストを取得する．

- `putMainText(idname, text)`: 計算結果や AI の回答を特定のコンポーネントへ出力する．
- `exeMainText(idname)`: 取得したテキストに基づき，計算や推論，レンダリングを実行する．

4　応用事例：微分演習ドリルの構築

再構築された Palette システムを用いることで，数式処理エンジン（CAS）と大規模言語モデル（LLM）を組み合わせた高度な数学教材を短期間で構築可能となった．本セクションでは，具体的な API の利用方法と，段階的な教材の高度化プロセスについて述べる．

4.1　基本的な正誤判定の実装（事例 1）

まず，テキストエリア，ボタン，Algebrite（CAS），TeX 表示コンポーネントを組み合わせた基本的な微分ドリルを構築した． 学習者の解答（‘sans‘）と正解（‘cans‘）の等価性を，CAS を用いて判定するロジックは以下の通りである．

```
// 答え合わせボタンの処理
var sans = PaletteComponent.getMainText("sans"); // 学習者の入力
var cans = PaletteComponent.getMainText("cans"); // 正解
// CAS(Algebrite)で差が0になるか判定
PaletteComponent.putMainText("alge", "simplify((" + sans + ") - (" + cans +
    "))");
PaletteComponent.exeMainText("alge");
var result = PaletteComponent.getMainText("alge");

if (result == 0) {
    alert("正解");
} else {
    alert("不正解");
}
```

ソースコード 1: 微分ドリルの正誤判定と LLM 連携 1

この手法では，単なる文字列比較ではなく，数学的な簡約化（‘simplify‘）を用いることで，解答の表記揺れ（例：$2x+1$ と $1+2x$）を許容した正確な判定が可能となる．このような表記揺れを許容する判定は，数式の本質理解を評価するために有用である．

4.2　乱数による問題の動的生成（事例 2）

教材の反復学習効果を高めるため，JavaScript の乱数生成機能を用いて問題を動的に作成する機能を実装した．

```
// 問題出題ボタンの処理
var m = 2, n = 10;
var a = Math.floor(Math.random() * (n - m + 1)) + m;
var b = Math.floor(Math.random() * (n - m + 1)) + m;
var c = Math.floor(Math.random() * (n - m + 1)) + m;

// 問題文の生成とTeX表示
var p = a + "x^2+" + b + "x+" + c + " を微分しなさい";
PaletteComponent.putMainText("tex", p);
PaletteComponent.exeMainText("tex");
```

```
11
12 // 内部的な正解データの更新
13 PaletteComponent.putMainText("cans", (2 * a) + "x+" + b);
```

ソースコード 2: 微分ドリルの正誤判定と LLM 連携 2

これにより，教員が静的な問題を多数用意することなく，無限に演習パターンを提供できる環境が構築された．毎回異なる問題を生成できるため，反復練習と自律学習に適している．

4.3 LLM による動的な解説生成（事例 3）

学習者が誤答した際，単に「不正解」と表示するだけでなく，LLM（Gemini）を用いて個別の解説を提供する機能を統合した．

```
1  var sans=PaletteComponent.getMainText("sans");
2  var cans=PaletteComponent.getMainText("cans");
3  PaletteComponent.putMainText("alge","simplify(("+sans+")-("+cans+"))");
4  var result = PaletteComponent.getMainText("alge");
5  var ca;
6  if (result==0) {
7    alert("正解");
8  // 誤答時の処理（事例1のelse節を拡張）
9  } else {
10     var ca = confirm("問題の解き方の解説が必要ですか?");
11     if (ca) {
12         // 問題文をLLMへのプロンプトに加工
13         var p2 = PaletteComponent.getMainText("tex")
14                  .replace('\\text{ を微分しなさい}', '微分しなさい．回答は $ で
                        囲んだTeX形式にしてください． ');
15         PaletteComponent.putMainText("LLM", p2);
16         PaletteComponent.exeMainText("LLM");  // LLMへの問い合わせ実行
17     }
18 }
```

ソースコード 3: 微分ドリルの正誤判定と LLM 連携 3

ここでは，既存のコンポーネント（‘tex‘）のテキストを取得・加工して LLM コンポーネントへ渡すという，コンポーネント間連携が実現されている．これにより，学習者は自身の状況に応じた即時的なフィードバックを，数式（KaTeX）を含んだ読みやすい形式で受け取ることができる．このような誤答直後の説明提示は即時フィードバックとして有効である．

Palette システムのシステム構成図を図 1に示す．本システムのアーキテクチャは，教育現場での導入の容易さを考慮し，計算リソースをサーバー側に依存しないクライアントサイド実行形式を基本としている．図 1に示す通り，システムの核となるのは PaletteComponent API であり，getMainText，putMainText，exeMainText という 3 つの主要メソッドを介して，UI 要素，数式処理エンジン，および LLM の間のデータフローを制御する．具体的な処理フローとして，学習者が入力した解答の数学的検証は，ブラウザ上で動作する Algebrite または Nerdamer によってリアルタイムに行われる．一方で，誤答に対する動的な解説生成が必要な場合は，API を通じて外部の Gemini (LLM) と連携する構成をとっている．最終的な出力は，KaTeX および Marked.js を介することで，数式とリッチテキストが混在した数学的に適切な表記としてレンダリングされる．

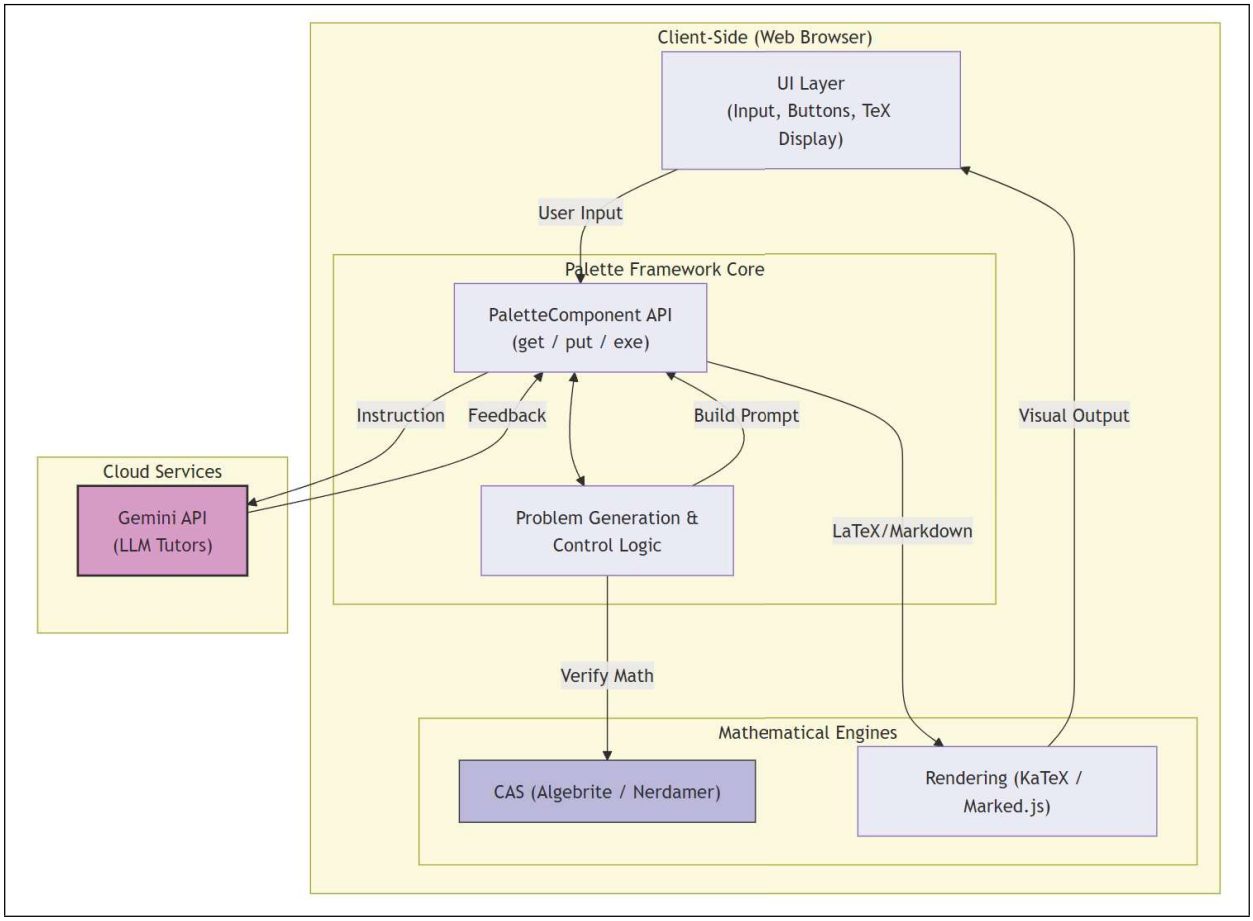

図 1: Palette システムのシステム構成図

5　考察：AI による開発の利点と課題

5.1　AI ツールの導入による開発パラダイムの変化

本研究における Palette システムの再構築プロセスは，AI が単なるコード補完ツール（アシスタント）から，開発タスクそのものを担う自律的な「パートナー」へと進化したことを示唆している．開発者の約 82%が日常的に AI ツールを利用しているという現状は，AI がワークフローに不可欠な存在であることを裏付けている．特に今回使用した Cursor は，プロジェクト全体の文脈を理解する能力に長けており，複雑なリファクタリング作業において強力な支援となった．

5.2　開発における利点：高速化と民主化

AI ツールの活用は，以下の点で大きな利点をもたらした．

- **プロトタイピングの超高速化**: 自然言語による指示からボイラープレートコードを自動生成することで，アイデアを即座に形にする PoC（概念実証）のサイクルが短縮された．

- **開発の民主化**: 高度な専門知識が不足している領域でも，AI の支援によって実装が可能となり，開発のハードルが下がった．
- **ドキュメント生成の効率化**: 難解なコードの解説やドキュメントの自動作成機能は，開発の透明性を高める上で非常に有用であった．

本研究においても，UI コンポーネントの雛形生成や API 接続コードの試作では高い効率化が見られた．

5.3 「生産性の矛盾」とデバッグコスト

一方で，AI の導入がすべての開発作業を一様に効率化するわけではない点も確認された．特に，既存コードベースの構造理解を要する修正やデバッグでは，AI の提案をそのまま適用できない場面があり，かえって修正コストが増加する場合があった．本研究でも，AI が誤った方向の修正案を繰り返し提示した際には，人手で設計意図を再整理した上で作業をやり直す必要があった．したがって，AI は開発を代替する存在というより，限定条件の下で有効に機能する支援手段として位置づけるのが妥当である．

5.4 信頼性とセキュリティの課題

AI ツールの高い導入率に対し，その出力に対する信頼性は依然として低いという「信頼のパラドックス」が存在する．

- **セキュリティ負債**: AI が生成したコードの約 40%に脆弱性が含まれるという指摘があり ([6]) ，SQL インジェクションなどのリスクに対する厳格なレビューが不可欠である
- **ハルシネーション（幻覚）**: AI が存在しない API や関数を生成するハルシネーションのリスクは，開発者を混乱させる要因となる．
- **スキルの空洞化（デスクキリング）**:AI への過度な依存により，特に若手開発者が基礎的なアルゴリズムや問題解決能力を深める機会を失う懸念も無視できない．

したがって，本システムのように学習支援機能と外部 API 連携を含む構成では，AI 生成コードに対する逐次的レビューが不可欠である．

5.5 結論としての展望

AI コーディングツールは，開発時間を削減しエラーを減少させる傾向にあるものの ，最終的な正確性やセキュリティの担保には依然として人間の手による厳格なチェックが必要である．今後は，開発者が「コードを書く人」から，AI が生成したコードを監督し，全体の設計を司る「アーキテクト」へと役割をシフトさせていくことが求められるだろう．

6　おわりに

6.1　本研究の総括

本研究では，複雑化した数学学習支援システム「Palette」を Cursor によって再構築し，CAS および LLM を統合した教材開発基盤を実現した．その結果，正誤判定，問題生成，誤答時の解説提示を備えた微分演習ドリルを比較的短期間で実装できることを示した．一方で，AI によるコード生成は初期実装や定型処理に有効である反面，複雑な依存関係を伴う修正や品質保証では人間による精査が不可欠であった．

以上より，AI-native IDE は教育用システム開発を加速する有力な手段であるが，その活用には設計判断とレビューを担う人間の関与が前提となる．

6.2　今後の課題

本研究で得られた知見を踏まえ，今後は以下の課題に取り組む必要がある．

- **未実装機能の拡充**: 現時点では一部の機能（compound コンポーネント等）の実装が完了しておらず，これらの再構成を早期に進める必要がある．
- **教材アーカイブの充実**: 今回開発した微分演習ドリル以外にも，線形代数や統計学など，多様な数学領域における教材作成例を追加し，システムの汎用性を検証する．
- **実験用ダッシュボードの開発**: 学習者の操作ログや LLM との対話履歴を収集・分析し，指導者が学習状況をリアルタイムで把握できる管理用ダッシュボードの機能を強化する．
- **AI との共生的な開発手法の確立**: 自律型エージェント（DEVIN 等）の台頭など，急速に進化する AI 技術をどのように開発プロセスへ取り込み，持続可能なシステム保守体制を構築していくか，その方法論を検討し続ける必要がある．

謝辞

本研究は JSPS 科研費 25K06706, 21K02752 の助成を受けている．

参　考　文　献

[1] Kitamoto, T., Kaneko, M., Noda, M., Kihara H. *E-Learning material creation system that utilizes existing teaching materials.* Proc. of The 29th Asian Technology Conference in Mathematics, Yogyakarta. 2024.

[2] Kitamoto, *T. Framework for building an E-Learning system using existing teaching materials.* Proc. of 14th MathUI Workshop 2023. 123-132, 2023.

[3] Stack Overflow. *2024 Developer Survey.* https://survey.stackoverflow.co/2024/ai (2026 年 3 月 11 日閲覧)

[4] Anysphere, Inc., *Cursor: The AI-native Code Editor.* https://www.cursor.com/ (2026 年 3 月 11 日閲覧)

[5] GitHub. *The state of the Octoverse: AI and the next generation of developers.* https://github.blog/news-insights/octoverse/ (2026 年 3 月 11 日閲覧)

[6] Pearce, H., et al. (2021). *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*, IEEE Symposium on Security and Privacy 2022, arXiv preprint arXiv:2108.09293.