論　文

生成AIを用いたC言語プログラミング学習のための助言システムの試作

若谷彰良[a], 前田利之[b]

[a] 甲南大学 知能情報学部
神戸市東灘区岡本８－９－１, 658-8501
[b] 阪南大学 経営情報学部
大阪府松原市天美東５－４－３３, 580-8502

(受理日 2023 年 11 月 27 日)

概 要

LLM(大規模言語モデル) を用いた生成 AI の技術の進歩はめざましく, プログラムのコード生成の技術をプログラミング教育に応用する試みは注目されている. 本稿では, OpenAI の API を用いて, エラーを含む C 言語のプログラムとエラーメッセージを LLM に入力として取り込み, 適切な助言を出力するバーチャル TA (Teaching Assistant) システムを初心者レベルの学習者向けに試作し, その適確性を評価する. プログラミングエラーの中で, 構文エラーと意味エラーについては概ね適確な助言が生成されており, 学習者にとって自力で問題解決するのに十分であるが, 論理エラーに対しては, いくつかのケースで一般的な説明に終始する程度の助言だけであったが, 適確な助言を生成できている場合もあり, 全般的には提案システムの有効性は確認できた.

キーワード： ChatGPT, OpenAI, API, GPT, プログラミング教育, 授業補助

1　はじめに

近年, LLM (大規模言語モデル) を用いた生成 AI の技術の進歩はめざましく, 新しいシステムが次々と発表されている. OpenAI の ChatGPT [1] や Google の Bard [2] のような対話型 AI や生成 AI ではテキスト生成や自然な対話などが一定の精度で行え, 1 億人を超えるユーザの獲得に成功しているが, 間違った回答を行うハルシネーションの問題や, ChatGPT では 2021 年までの情報のみで学習されているなどの課題も残っている. しかし, プログラムのコード生成に関する技術では完成度が高く, OpenAI の GPT-4 をベースにした Code Interpreter [3] では, “csv に格納された気温と売り上げのファイルから scikit learn を用いて回帰分析する Python プログラムを作成せよ” という文章から Python プログラムを生成するだけでなく, 適当なデータを与えれば実行まで行うことができるようになっている. すなわち, プログラミング言語を用いたプログラムを作成する, 解釈するといった機能に関しては, 生成 AI の完成度はかなり高度なレベルにまで達しているといえる.

一方, IT や AI における人材不足は深刻であり, 世間の要請にあった技術者の養成は喫緊の課題となっている. IT や AI の技術者の養成において, プログラミング技術の修得は必須項目の一つとなっている. プログラミング学習においては, 自動的にプログラムが生成されることよりも, 自分で書いたプログラムをどのように修正すれば良いかを自らが学ぶことがより重要である. その際, 初学者のプログラミング学習では, つまずいたポイントに対する TA (Teaching Assistant) からの助言が有効であり, 正しいプログラムを提示されただけでは学習につながらないことが知られている. 例えば, 文献 [4] において, 模範解答を即座に提供することは教育学的な観点から問題があると述べられており, また, LLM を微調整して正解を与えず, 代わりに生徒が解答にたどり着くのを手助けすることが提案されていることなどから, LLM はコードの経験は豊富だが, 優れた教育法についてはあまり理解していないと結論づけられている.

AI を用いてプログラミング学習を効率化する試みは既に開始されている. 文献 [5] において Kazemitabaar らは, OpenAI Codex を用いて初学者のプログラム作成にどの程度の有効性があるかを調べ, プログラムの完成率で 1.15 倍, 成績で 1.8 倍の向上を確認している. さらに, プログラム作成能力だけでなく, 事後に行った確認テストの成績向上も確認されている. また, 文献 [6] において Leinonen らは, Python 言語におけるエラーメッセージに対する GPT-3 の解説が, そのエラー内容の理解に有用であり, temperature パラメータの調整が重要になっていることを示した. ただし, この研究で対象としている Python 言語はインタプリタで実行されており, プログラムの解釈時の情報及び実行時の情報の両方をもってエラーメッセージを生成できていることから, システムが出力する説明も適確であると考えられる. よって, タイプの異なるコンパイルエラーと実行時エラーが存在する C 言語のようなコンパイラ言語に対しては, この研究の結果だけではその有用性は判断できない.

そこで, 本稿では, 高度な生成性能を持つ GPT-4 もしくは GPT-3.5-Turbo のような LLM を用い, プログラミングエラーを含むプログラムに対して, 修正済のコードを提示することなく, 助言だけをあたえるようなプログラミング学習サポートシステム (バーチャル TA (Teaching Assistant) システム) の試作を行なう. さらに, コンパイルエラーと実行時エラーが含まれる C 言語を対象として, 典型的なエラーメッセージに対して本システムが出す助言の適確性を評価する.

2 プログラミングエラー

C 言語のプログラムの作成において, プログラムの間違いであるバグの発生要因は, 構文エラー (字句エラーを含む), 意味エラー, 論理エラーの 3 個の要因に大別できる [7]. 最初の 2 個はコンパイル時に発見されることが多いのでコンパイルエラーと呼ばれ, 3 個目は実行時に発見されることが多いので実行時エラーとも呼ばれる. 構文エラーは文法的な間違いによるもので, `int` と `integer` の混同や, `elseif` と `elif` の混同等のキーワードの書き間違い, セミコロンやカンマの書き忘れ, 演算子の使用間違い, カッコの種類の使用間違い等, 文法や字句の覚え間違いに起因するものである. 初学者においては, プログラミング言語自体になれていないので, 数学の書式をまねて間違うなど, 頻発するエラーの要因となっている.

2 番目の要因である意味エラーは, 文字もしくは字句の並び方である構文としては正しいが, プログラムの意味としては正しくない場合に起こる. 例えば, 変数 `xx` を `int` 型と `double` 型の両方で宣言することは構文的には可能であるが, 1 個の変数が 2 個の型を持てないので意味的には正しくない. また,

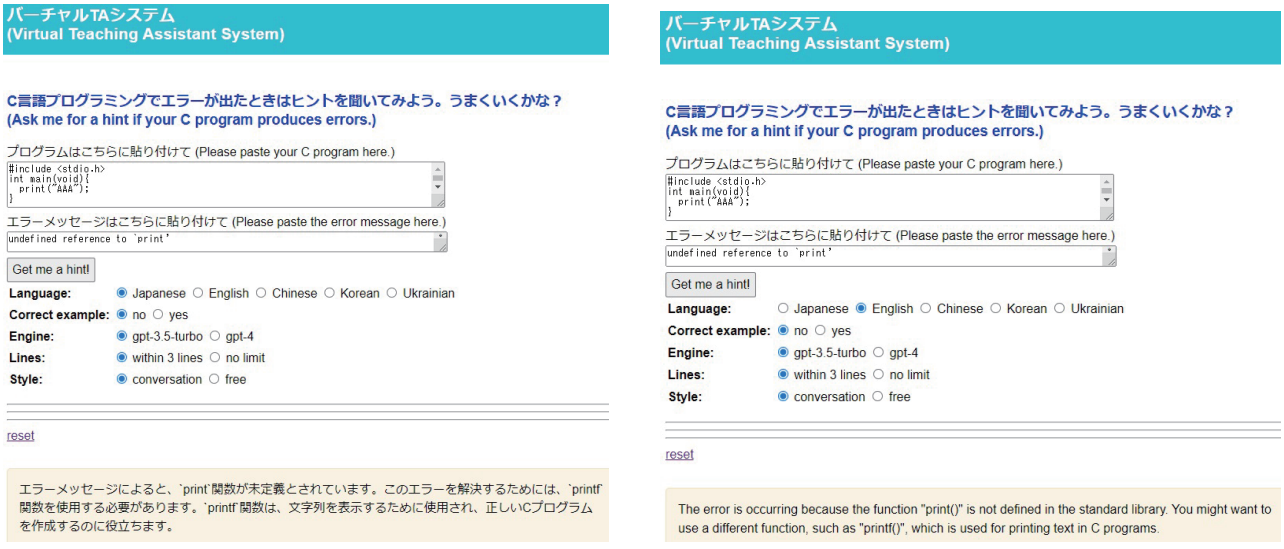
関数 `func` の引数が `int` 型の変数 1 個で定義しているのに, その関数 `func` を 2 個の引数を与えて呼び出すことはできないが, 構文的にはエラーは生じない. その他, C 言語において, `unsigned float` のような許されていないキーワードの組み合せを用いたり, 定義がされていないラベルへの分岐や, ポインタ変数でない変数によるポインタ参照など, 諸々の要因が含まれる.

3 番目の要因である論理エラーは, 上記の構文エラー及び意味エラーと異なり, 厳密には実行することで初めてエラーであることが分るものである. 例えば, サイズが 10 の配列 `x` に対し, `i=10;` とした後, `x[i+3]` で配列要素にアクセスした場合, メモリーアドレスエラーになる. また, `i=10;while(i>1)i++;` というプログラムの場合, 変数 `i` は 10 から無限に大きくなり無限ループとなるか, 変数の値がオーバーフローする. これら以外にも, プログラムを見ただけではエラーが発生することを確実には予見できない場合があり, これらのエラーを論理エラーと呼ぶ.

このように初学者のプログラミング学習においては様々なエラーが生じ, 初学者本人では解決できない場合がある. そこで, TA の助言があれば初学者にとってもそのエラー内容を容易に理解でき, プログラミング言語を修得するための学習効率が向上すると予想される.

3 バーチャル TA システム

これまでに述べたように, 初学者のプログラミング学習においては, 自分で作ったプログラムの間違いに対して適確な助言を与える TA の存在が重要である. そこで, そのような TA の助言を OpenAI の API を用いて表示するバーチャル TA システム (以下, VTA システム) を試作する. 図 1 にシステムの画面キャプチャを示す. 左側が日本語での出力画面で, 右側は英語での出力画面である.



(a) Japanese (b) English

図 1: Virtual TA system

現状の VTA システムは C 言語専用であり, 元となるプログラムとそのプログラムをコンパイル及

び実行した際のメッセージを入力するエリアがあり, “Get me a hint” のボタンをクリックすると適確な助言が表示される.

用いる LLM (大規模言語モデル) は, GPT-4 もしくは GPT-3.5-Turbo のいずれかであり, それを選択でき, 出力言語の選択 (日本語, 英語, 中国語, 韓国語, ウクライナ語) も可能である. また, 助言の中に正しいプログラムの書き方の例を示すこと (Correct example) も可能であるが, プログラミング学習の効果からは見せないほうが良いとされているので, デフォルトではこの機能は選択されていない. さらに, 助言の長さも短かい方が良いので, 助言の長さ (Lines) を 2-3 行のパターンと長さを制限しないことの 2 パターンを用意し, 2-3 行の助言をデフォルトとしている.

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "https://api.openai.com/v1/chat/completions");
...
$data = array('model' => 'gpt-4');
...
$data["messages"] = array();
$message3='appropriate prompt'
$data["messages"][] = array('role' => 'user', 'content' => $message3);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
$result = curl_exec($ch);
$response = json_decode($result, true);
curl_close($ch);
```

図 2: CURL を用いた PHP による OpenAI API プログラム (一部を省略)

本システムは, OS が Ubuntu 20.04 LTS のコンピュータ上に PHP 7.4.3 が利用できる Apache/2.4.41 の web サーバを動作させ, JQuery 3.5.1 と OpenAI API を利用する PHP プログラムとして作成している. 図 2 に CURL を用いた PHP による OpenAI API プログラムを示す. 用いる LLM は GPT-4 だけでなく GPT-3.5-Turbo に変えることも可能であり, プログラム中の`$message3` に適切なプロンプトの文字列を設定する.

```
My C program is as follows:
#include <stdio.h>
int main(void){
print("AAA");
}
I got the following error message.
undefined reference to ‘print’
Show me a hint to get a correct program in Japanese, in conversation sentence
    style, within 3 lines, but, do not show the correct C program.
```

図 3: プロンプトの例

OpenAI API はやり取りするデータ量により料金が決定される. データ量は token という単位でカウントされるが, 英語であれば概ね 1 単語で 1 token となるが, 日本であれば 1 文字で数 token を消費することが知られている. データ量を抑えるためにプロンプトは英語で送信することとし, “in Japanese”

のように, プロンプト中に出力言語を指示することで出力言語を設定する.

図3に, 図1の左図の出力を得るためのプロンプトを示す. ブラウザで設定するオプションに従った文言をプロンプトに付加しているが, “in conversation style” は, 回答を話し言葉で出力することを指示している. このオプションを用いる理由は, 将来的に回答を文字だけなく音声で出力することを想定しており, より実際のTAに近い使い方を目指している.

4 試用と評価

4.1 評価方法

既に述べたように, プログラミングエラーに対する助言を行なうWebアプリケーションをOpenAI APIを用いて実装している. このシステムに, コンパイルエラーである構文エラーと意味エラー, 及び実行時エラーである構文エラーを含んだC言語のプログラムと, そのエラーメッセージを与え, 出力される助言が適確かどうかを評価する. エラーメッセージは, Ubuntu 20.04 LTSのOSが動作してるコンピュータ上の, GCCコンパイラ (9.4.0) が出力するものを利用する.

また, C言語のプログラムは各エラー毎に5パターン用意し, 同じ設定で助言の取得を3回行ない, それぞれの評価を行った. オプションとしては, LLMにGPT-3.5-Turbo[1], 出力の行数はFree(2-3行の指定はせず), 正解となる修正後のC言語プログラムを出力させないようにし, temperatureオプションはデフォルトのまま使用した.

各エラーに対して出力された助言の評価を以下のサブセクションの表1, 表2, 表3に示す. 評価は著者らによる主観評価で行なう. 評価は5段階とし, ◎: 行番号もしくは変数名などを使って適確に説明, ○: 行番号もしくは変数名の記載はないが適確に説明, ◉: 適確に説明しているが修正済プログラムも提示, △: 正しい説明だが一般論のみ, ×: 間違った説明, のいずれかとする.

4.2 構文エラー

表1: 構文エラーに対する助言の評価

Error	Factor	1	2	3
A. Syntax	1. missing semicolon	◎	◎	○
	2. illegal operator	◎	◎	◎
	3. missing parenthesis	◉	◎	◎
	4. wrong keyword	◎	◎	◎
	5. missing operator	◎	◎	◎

[1] この実験の実施時には, まだGPT-4が利用できなかった.

表 1 に示すように, 構文エラーを含むプログラムは, 1. missing semicolon, 2. illegal operator, 3. missing parenthesis, 4. wrong keyword, 5. missing operator の 5 個を例として採用した. 1. missing semicolon では, 実行文の最後のセミコロンを書き忘れたもので, 2. illegal operator は, `i=j++k;` という間違った演算子の使用法を含むものである. 3. missing parenthesis は, if 文のブロックの閉括弧を書き忘れたもので, 4. wrong keyword は, `int` の代わりに `integer` で変数宣言するものであり, 5. missing operator は, `i=(j+k)(j-k);` のように一部の演算子を書き忘れている代入文を含むプログラムである.

図 4 は 1. missing semicolon の実行例の一つであり, プログラムの行番号を含めて適確な助言が出力できている.

(program)

```
1 #include <stdio.h>
2 int main(void){
3   int i,j,k;
4   i=j+k
5 }
```

(error message)

```
1 5:8: error: expected ‘ ’; before
    ‘ ’} token
2     5 | i=j+k
3       | ^
4       | ;
5     6 | }
6       | ~
```

(comment) エラーメッセージには、5 行目のカッコの前にセミコロンが必要だと言われています。
プログラムの文法を理解し、文にセミコロンが必要なことを覚えておく必要があります。

図 4: 実行例 1 (missing semicolon)

実行例 1 に示した例だけでなく, 構文エラーに対するエラーメッセージは行番号を含んでいる場合が多いので, 助言に行番号や変数名などを含むことが多く, 概ね適確な助言 (◎) が出力できている. しかし, 1. missing semicolon において, 下記のように, 場所を明示しない助言を出力する場合があった.

ヒント：セミコロンを追加して演算式を修正しましょう

LLM による文章生成は非決定的な動作をするので, 十分な助言が得られない場合は, “聞き返す” ような機能をつけて不十分な助言の補完をする必要があると考えられる.

4.3 意味エラー

表 2 に示すように, 意味エラーを含むプログラムは, 1. no type declaration, 2. type declaration error, 3. attribute error, 4. function error 1, 5. function error 2 の 5 個を例として採用した. 1. no type declaration は, プログラム内の一部の変数の宣言を書き忘れたもので, 2. type declaration error は, プログラム内の一部の変数が複数の型で宣言されたものである. 3. attribute error は, `unsigned float` という許されていないキーワードの組合せを含むもので, 4. function error 1 は, 引数が 1 個の関数を 2 個の引数で呼び出すプログラムで, 5. function error 2 は, 返り値を持たない関数を `i=func(k);` で呼び出すプログラムである.

図 5 は 1. no type declaration の実行例の一つであり, 宣言がされていない変数名を明確にした上で, 適確な助言が出力できている.

表 2: 意味エラーに対する助言の評価

Error	Factor	1	2	3
B. Semantic	1. no type declaration	◎	◎	◎
	2. type declaration error	◎	◎	◎
	3. attribute error	◎	◎	◎
	4. function error 1	◎	◎	◎
	5. function error 2	◉	◉	◎

実行例 2 に示した例だけでなく, 意味エラーに対するエラーメッセージは対象となる変数名, 関数名や行番号を含んでいる場合が多いので, 助言に十分な情報を付加して, 適確な助言 (◎) が出力できている. ただし, 修正済プログラムの表示は拒否する設定にしているが, 数パターンにおいて表示がされている. プロンプト内に必要な指示は明示はしているが, 設定を完璧にするために指示を強調するようなプロンプトの工夫が必要であると考えられる.

(program)

```
1 #include <stdio.h>
2 int main(void){
3   int i,j;
4   i=j+k;
5 }
```

(error message)

```
1 5:7: error: ‘ ’k undeclared (first
    use in this function)
2     5 | i=j+k;
3       | ^
4 5:7: note: each undeclared
    identifier is reported only once
    for each function it appears in
```

(comment) このエラーメッセージは、変数”k”が定義されていないという意味です。そのため、予め”k”を宣言する必要があります。

図 5: 実行例 2 (no type declaration)

4.4 論理エラー

表 3 に示すように, 論理エラーを含むプログラムは, 1. out of region access 1, 2. out of region access 2, 3. zero division 1, 4. zero division 2, 5. deep recursive call の 5 個を例として採用した. 1. out of region access 1 は, 宣言した配列サイズを越えるインデックスをもった配列要素の参照を行なうプログラムで, 2. out of region access 2 は, 1 と同様に宣言した配列サイズを越えるインデックスをもった配列要素の参照を行なうプログラムであるが, インデックスの計算がやや複雑なものである. いずれのエラーメッセージも “Segmentation fault” のみである. 3. zero division 1 は, 値が 10 の変数 `i` で `i/(i-10)` の値を

用いるプログラムで, 4. zero division 2 は, 値が 10 の変数 i で i/(i-10) と i/(i+10) の両方の値を用いるプログラムであり, いずれのエラーメッセージも “浮動小数点例外” のみである. 5. deep recursive call は, 再帰呼出しをする関数の呼出しの深さが深く, スタック領域が枯渇するプログラムで, エラーメッセージは “Segmentation fault” のみである.

表 3: 論理エラーに対する助言の評価

Error	Factor	1	2	3
C. Logical	1. out of region access 1	◎	◎	◎
	2. out of region access 2	◎	◎	◎
	3. zero division 1	△	○	×
	4. zero division 2	△	◎	○
	5. deep recursive call	◉	◎	◎

図 6 は 5. deep recursive call の実行例の一つであり, 関数 func が再帰関数であること, 及びその再帰呼出しの回数が大きいことを理解した上で, その解決方法を含めて適確な助言が出力できている.

(program)

```
1 #include <stdio.h>
2 int func(int x){if(x==0) return 0;
      else return func(x-1)+2;}
3 int main(void){
4   int i=1000000;
5   printf("\%d\n",func(i));
6 }
```

(error message)

```
1 Segmentation fault コアダンプ ()
```

(comment) セグメンテーションフォルトは、プログラムが無効なメモリを参照しようとしたときに発生するエラーです。このエラーの原因は、再帰関数 ‘func()‘ の引数 ‘x‘ の値が大きすぎるためです。解決策としては、再帰の深さを減らすか、メモ化を使用することが考えられます。
再帰の深さを減らすためには、再帰呼び出しの代わりにループを使用することができます。以下は、‘func()‘ をループで実装した例です。

図 6: 実行例 3 (deep recursive call)

構文エラーや意味エラーでは適確な助言を与えられていたが, 論理エラーに対しては十分でないケースがある. out of region の場合は概ね適確 (◎) で, 例えば, 下記のように,Segmentation fault のエラーメッセージを理解し, 配列サイズに関してそれを確認するようにとの助言を生成している.

> 配列 ”d” と ”c” のサイズは 1000 であり、配列のインデックスは 0 から始まります。したがって、インデックスの範囲は 0 から 999 です。配列要素にアクセスする際に、これらの範囲内に収まっていることを確認してください。

学習者は, プログラムの何行目を見ればいいかまでは分らないが, 助言で言及された配列のインデックスに注目することは理解できる. また, 前述の通り, deep recursive call に対しても適確な助言を生成できている.

一方, zero division に対する助言は一般的な説明のみ (△) や間違い (×) の場合もあり, 適確な助言を生成できていないケースがある. 例えば, 4-3 の間違いの助言 (×) は下記のようになっており, 除算の分母に注目をしてはいるものの, 数学的および論理的に意図が理解できない説明になっている.

> 問題の原因を正しく理解するために、変数 ‘i‘ の値 ‘10‘で除算を行っています。除算の分母としては ‘i‘ の値から ‘10‘を引くため、 ‘i‘ の値は必ず ‘10‘でなければなりません。しかし、あなたのコードでは ‘i‘ の値を ‘10‘で初期化しています。

また, 4-1 の一般的な説明のみの助言 (△) は下記のようになっており, プログラム中のどの部分に注目してよいかが学習者には分かり難くなっている.

> このエラーメッセージは、プログラムが浮動小数点例外を発生させたことを示しています。ヒント：プログラムが 0 で割り算をしているため、浮動小数点例外が発生しています。割る数に注意してください。

しかし, 除算を含む実行文を示して, その部分を見直すように適確に助言する場合もあるので, 複数回助言を求めるようにすれば適確な助言を得られる可能性はあるが, 学習者にとっても負担が大きく, プロンプトの改良などの対策を今後検討していく.

5 おわりに

エラーを含む C 言語のプログラムとエラーメッセージを LLM に入力として取り込み, 適切な助言を出力するバーチャル TA システムを, OpenAI の API を用いて初心者レベルの学習者向けに試作し, その適確性を評価した.

プログラミングエラーの中で, 構文エラーと意味エラーについては概ね適確な助言が生成されており, 学習者にとって自力で問題解決するのに十分である. しかし, C 言語のようなコンパイラ言語ではコンパイル時の情報, 例えば, プログラム上の行番号と機械語命令の命令番号の対応やプログラム内の変数とそのメモリ上のアドレス等は実行時には保持されておらず, 実行時エラーの際にはプログラム上の情報を用いたエラーメッセージは生成しにくい. よって実行時エラーの一つである論理エラーに対しては, いくつかのケースで一般的な説明に終始する程度の助言だけのものがあったが, 適確な助言を生成できている場合もあり, 全般的には提案システムの有効性は確認できた. また, 不十分な助言の場合に “助言を聞き直す” 機能を追加することで, 有用性を改善することが期待できる.

今後は, プログラムエラーを含むプログラム例と評価項目をさらに増やして評価をより精緻なものとするとともに, 他の言語, 例えば Java 言語などにも応用することなどを通して本システムの有効性を確認し, ソフトウェア教育へ貢献していきたい.

謝辞

本研究の一部は JSPS 科学研究費 (基盤研究 (C) 18K02920, 基盤研究 (C) 19K03018, 基盤研究 (C) 23K02671) 及び私立大学等経常費補助金特別補助「大学間連携等による共同研究」による.

参考文献

[1] https://chat.openai.com/ (as of 2023/10/18)

[2] https://bard.google.com/ (as of 2023/10/18)

[3] https://chat.openai.com/?model=gpt-4-code-interpreter (as of 2023/10/18)

[4] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpaa and J. Sorva, “Exploring the Responses of Large Language Models to Beginner Programmers’ Help Requests,” https://arxiv.org/pdf/2306.05715.pdf, 13 pages, 2023. (As of 2023/10/18)

[5] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop and T. Grossman, “Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming”, in *Proc. the 2023 CHI Conference on Human Factors in Computing Systems*, DOI:10.1145/3544548.3580919, 23 pages, 2023.

[6] J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather and B. A. Becker, “Using Large Language Models to Enhance Programming Error Messages,” in *Proc the 54th ACM Technical Symposium on Computer Science Education*, DOI: 10.1145/3545945.3569770, pp. 563-569, 2023.

[7] A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd ed.).* Pearson Education, 2006.