修士学位論文

題目

構文誤りを含むソースコードを評価可能な指標の比較
―翻訳ベースのコード生成を前提として―

指導教員

楠本 真二 教授

報告者

髙市 陸

令和 5 年 2 月 1 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻

令和 4 年度 修士学位論文

構文誤りを含むソースコードを評価可能な指標の比較
—翻訳ベースのコード生成を前提として—

髙市 陸

内容梗概

近年，ソースコードを全自動で生成する技術であるコード生成の研究が盛んに行われている．コード生成は発展途上の技術であり，構文誤りを含むソースコードが生成されることも少なくない．そのような誤りを含むソースコードに対しては，抽象構文木やプログラム依存グラフを利用するソースコード用の評価指標は適用できない．一方，自然言語処理で用いられる評価指標ならば構文誤りの有無にかかわらず適用可能である．それらの自然言語処理で用いられる評価指標のうち，どの指標が自動生成されたソースコードの評価に適しているかは明らかでない．そこで本研究では，構文誤りを含む自動生成コードに対しても適用可能で評価に適した指標を明らかにする．生成コードの修正容易性と相関の強い評価値を出力する指標は，生成コードの評価に適していると考えた．なぜなら，要求を満たすような正しいコードの生成は現状のコード生成では困難であり，実際に利用する際，開発者による生成コードの修正が必要だからである．生成コードを要求を満たすようなコードに修正する被験者実験を実施し，修正容易性を生成コードの修正に要した時間として測定した．そして，実験により得られた修正時間と評価指標による生成コードの評価値との相関をとり，生成コードの評価に適した指標を調査した．調査の結果，生成コードの評価に比較的適している既存の評価指標は chrF であることが明らかになった．また，既存の評価指標が持つ特徴を組み合わせることでより良い評価指標の作成を試みた．

主な用語

評価指標，コード生成，深層学習

目次

図目次

表目次

1　はじめに

近年, コード生成というソースコードを全自動で生成する技術の研究が盛んに行われている [1,2,3,4]. コード生成手法の研究は発展途上で，構文的に間違っているソースコードが生成されることも少なくない．近年提案されたモデルによって生成されたコードであっても 7％ から 90％ は構文的に正しくなかったという報告がある [5].

コード生成の研究では，評価指標を用いて生成されたコードの正しさを評価する．ソースコードの構造的な情報を利用して評価を行う指標がいくつか提案されている [6, 7]．そのようなソースコード用の評価指標は，対象のコードが構文的に正しいことを前提として抽象構文木やプログラム依存グラフを用いている．評価対象のコードが構文的に間違っている場合，抽象構文木やプログラム依存グラフは作成できない．そのため，構文的な正しさを前提としているソースコード用の指標は生成コードの評価に使えないことがある．

ソースコード用の評価指標ではなく，自然言語処理で用いられる指標がソースコードの評価に使われている場合もある [8]．このような自然言語処理で用いられる評価指標は構文誤りを含むコードに対しても適用可能なため，ソースコード用の評価指標よりも生成コードの評価に適している可能性がある．このような構文誤りを含むコードでも評価可能な指標のうち，どの指標が自動生成されたコードの評価に適しているかは明らかでない．

そこで本研究では，構文誤りを含む自動生成コードに対しても適用可能で評価に適した指標を明らかにする．特に翻訳ベースのコード生成タスクに着目し，自然言語で記述された要求から自動的に生成されたコードの評価に適した指標について調査する．

現状，自然言語で記述された要求を入力とするコード生成では，要求を満たすコードの生成は困難である [5]．コード生成を利用する際，開発者は生成されたコードを要求を満たすように修正する必要があるので，生成されたコードは要求を満たすコードへの修正が容易であることが望ましい．そのため，自動生成されたコードの評価指標による評価値と修正容易性の間に強い相関があるとき，その評価指標が生成コードの評価に適していると判断できると考えた．

生成されたコードから要求を満たすコードへの修正容易性を測る被験者実験を行った．生成コードの修正容易性は，そのコードを開発者が修正するために要した時間の短さによって測定した．その結果，生成されたコードの評価に比較的適している既存の評価指標は chrF [9] であることが明らかになった．また，既存の評価指標が持つ特徴を組み合わせることでより良い評価指標の作成を試みた．

以降，2 節では本研究で設定した Research Question についての説明を行い，3 節で本研究の研究動機と評価指標について述べる．4 節で Research Question に答えるために行った実験の内容と結果および考察について述べ，5 節で妥当性の脅威について述べる．6 節で本研究の関連研究を示し，最後に 7

節で本研究のまとめと今後の課題について述べる．

2 Research Questions

コード生成によって生成されたコードの評価に適した評価指標を明らかにするために，以下 3 つの Research Question（RQ）を設定した．

RQ1: 修正時間を用いて測定する生成コードの修正容易性と関連の強い評価指標はどれか？

生成されたコードは要求を満たすコードへの修正が容易であることが望ましい．そのため生成コードの修正容易性と関連の強い指標が生成コードの評価に適している．生成コードの修正容易性は，開発者による修正に要した時間で測定できる．生成コードの修正容易性と関連の強い評価指標を示すために，構文誤りを含むコードに対しても適用可能な 5 つの評価指標を対象として，修正時間と各評価指標の評価値の相関を調査する．最も相関が強い評価指標が生成コードを適切に評価できるとみなす．

RQ2: 生成コードの評価に適した評価指標はモデルごとに異なるか？

あるモデルが生成するコードの評価に適した指標と，別のモデルが生成するコードの評価に適した指標は異なる可能性がある．ある評価指標が特定のモデルに対しては適しているが別のモデルには適していない場合，そのような評価指標はモデル間の性能比較には適さない．モデル間の比較が可能な評価指標を示すために，2 つの異なるモデルの生成コードの評価に適した評価指標を調査する．2 つのモデルで共に修正容易性と関連の強い評価指標が存在すれば，その評価指標がモデル間の比較に適していると言える．

RQ3: どのような特徴を持つ評価指標が生成コードの評価に適しているか？

各評価指標は，評価対象のどのような点に着目して評価するかが異なっている．着目する点の違いは，その評価指標の特徴の違いと言える．生成コードの評価に適した評価指標の持つ特徴を調べることで，より適した評価指標の提案が可能となる．RQ1 の回答となる生成コードの評価に適した評価指標よりもさらに良い評価指標の提案のために，対象となる評価指標の各特徴を組み合わせた新たな評価指標を提案する．また，提案する評価指標がモデル間の比較に適しているかどうかも調査する．

3 準備

3.1 コード生成

コード生成とは，ソースコードを全自動で生成する技術である．コード生成手法は，入力として与える要求の形式とコード生成手段の 2 つの組み合わせで分類できる．入力の形式には自然言語の文章 [5]，DSL [10]，入出力例 [11] 等がある．手段には，翻訳ベース [5]，探索ベース [12] 等がある．本研究では特に，自然言語の文章で書かれた要求を入力とする翻訳ベースのコード生成手法に着目する．

コード生成手法の研究は発展途上で，構文的に間違っているソースコードが生成されることも少なくない．Liu ら [5] の報告によると，近年提案されたコード生成モデルである SNM [3] では 7 %，Coarse-to-Fine [4] では 90 % もの生成コードが構文誤りを含んでいた．

3.2 評価指標

生成コードのための評価指標は大きく以下の 2 つに分けられる．

- コードの構造的な類似に基づく評価指標
- トークンの一致やその並びの類似に基づく評価指標

コードの構造的な類似に基づく評価指標

ソースコード用の評価指標がいくつか提案されている [6, 7]．これらの評価指標は，コードが構文的に正しいことを前提として抽象構文木やプログラム依存グラフを用いている．構文誤りを含むコードの抽象構文木やプログラム依存グラフは作成できない．

トークンの一致やその並びの類似に基づく評価指標

生成コードの評価には，ソースコード用の指標ではなく，このような自然言語処理で用いられる評価指標が使われている場合もある [8]．このような評価指標は構文誤りを含むコードに対しても適用可能なため，ソースコード用の指標よりも生成コードの評価に適している可能性がある．例えば，図 1(a) のような生成コードを自然言語処理用の評価指標のひとつである BLEU で評価することを考える．評価のための正解コードとして図 1(b) を用いる．このとき，図 1(a) の生成コードは，未定義変数 `n`の使用や不正なトークン`<unk>`が含まれていることにより構文的に正しくない．そのため，ソースコード用の評価指標では評価できない．一方，自然言語処理用の評価指標では評価可能であり，BLEU による評価値は 0.890 となる．BLEU による評価は，1 に近いほど高いことを示しており，この生成コードは評価が高いことになる．3.2 節に示した 2 つのコードはともに FizzBuzz 問題を解くためのコードの例であ

```
def fizzbuzz(i):
    if n % 3 == 0 and n % 5 == 0:
        return "FizzBuzz"
    elif n % 3 == 0:
        return "Fizz"
    elif n % 5 == 0:
        return "Buzz"
    else:
        return <unk>
```

(a) 構文誤りを含む生成コード

```
def fizzbuzz(n):
    if n % 3 == 0 and n % 5 == 0:
        return "FizzBuzz"
    elif n % 3 == 0:
        return "Fizz"
    elif n % 5 == 0:
        return "Buzz"
    else:
        return str(n)
```

(b) 正解コード

図 1　FizzBuzz 問題を解くための構文誤りを含む生成コードとその正解コードの例

り，図 1(a) に示す生成コードは構文誤りを含むものの正解コードである図 1(b) にかなり近いことが分かる．このような生成コードなら，FizzBuzz が解けるようなコードへの変更は容易であり，BLEU による高い評価は妥当だと言える．

本研究では，自然言語処理用の評価指標を対象とする．このような評価指標は，2 つのテキストの字面の類似度を評価する．これらは，生成されたテキストの品質を評価するために考案されている．生成されたテキストの手動評価は高コスト [13] なため，機械的かつ自動的に低コストで評価できる評価指標が提案されている．評価指標の入力として生成コードと開発者が書いた正解コードをテキストとみなして与え，生成コードが正解コードにどれだけ類似しているかを出力している．

本研究で用いた評価指標とその特徴を表 1 に示す．以下では，各評価指標についてそれぞれ説明する．いずれも，評価値は 0 以上 1 以下の数値で表現され，1 に近いほど高い評価を意味する．

BLEU

BLEU（BiLingual Evaluation Understudy） [13] は，自然言語機械翻訳のために考えられた評価指標である．BLEU は，以下の 2 つの特徴を持つ．

- 生成コードが正解コードと比べて短すぎる場合には類似度が小さくなる

表 1　本研究で用いた評価指標とそれぞれの特徴

評価指標名	特徴
BLEU	トークン単位の n-gram の適合率を利用し，生成コードの長さを考慮した指標
STS	編集距離を利用した指標
ROUGE-L	最長共通部分列を利用した F 値ベースの指標
METEOR	トークンの並びと一致度を考慮した F 値ベースの指標
chrF	文字単位の n-gram を利用した F 値ベースの指標

```
S += b [ j ] + a [ i ]
```

(a) 生成コード X

```
S += a [ i ] + b [ j + 1 ]
```

(b) 正解コード Y

図 2　評価指標計算の説明用コード

- トークン単位の n-gram の適合率を利用している

以下では，生成コードのトークン列を X，正解コードのトークン列を Y とする．BLEU は，X と Y の n-gram の適合率 p_n と重み w_n，生成コードの長さに対するペナルティ BP を用いて式 (1) で表される．

$$\begin{aligned} \mathrm{BLEU}(X,Y) &= \mathrm{BP}\cdot\exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \\ \mathrm{BP} &= \min\left(1, \exp\left(1-\mathrm{length}(Y)/\,\mathrm{length}(X)\right)\right) \\ p_n &= \frac{\mathrm{length}\left(n{-}\mathrm{grams}(X)\cap n{-}\mathrm{grams}(Y)\right)}{\mathrm{length}\left(n{-}\mathrm{grams}(X)\right)} \end{aligned} \tag{1}$$

なお，$n{-}\mathrm{grams}(S)$ は列 S の n-gram の集合を表している．本研究では，$N=4$，$w_n = 1/N$ とする．これは，BLEU を提案した研究 [13] で用いられた設定である．

例えば，生成コードが図 2(a) で正解コードが図 2(b) のときの BLEU は次のように計算する．なお，図 2(a) および図 2(b) ではトークンの単位が分かりやすいようにトークン同士の間を広く空けている．まず，X に対する Y の n-gram の適合率を考える．$n=4$ のとき，X の 4-gram は 8 個あり，そのうち Y にも存在する 4-gram は“`a [ i ]`”のみで 1 個なので $p_4 = 1/8$ である．同様に $p_1 = 11/11$，$p_2 = 7/10$，$p_3 = 3/9$ となる．また，$\mathrm{length}(Y)=13$，$\mathrm{length}(X)=11$ より

$$\begin{aligned} \mathrm{BP} &= e^{-2/11} \\ \mathrm{BLEU}(X,Y) &= \mathrm{BP}\cdot\exp\left(\sum_{n=1}^{4}\frac{1}{4}\log p_n\right) \\ &\simeq 0.345 \end{aligned}$$

となる．

本研究では，正解コードが複数ある場合の BLEU は，正解コードの集合を Ys として式 (2) で算出する．

$$\mathrm{BLEU}(X, Ys) = \max_{Y\in Ys} \mathrm{BLEU}(X,Y) \tag{2}$$

STS

STS（STring Similarity） [6] は，編集距離を用いることが特徴の評価指標で式 (3) で表される．

$$\mathrm{STS}(X,Y) = 1 - \mathrm{NormarizedEditDistance}(X,Y) \tag{3}$$

編集距離 EditDistance [14] は，ある列 X を別の列 Y と同じ列にするための編集操作の最小回数で定義される．編集操作には要素の追加，削除，置換の 3 種類がある．値域が 0 以上 1 以下になるよう正規化された編集距離 NormarizedEditDistance の求め方を式 (4) に示す．なお，列 S の長さを $\mathrm{length}(S)$ で表している．

$$\mathrm{NormarizedEditDistance}\left(X,Y\right) = \frac{\mathrm{EditDistance}\left(X,Y\right)}{\max\left(\mathrm{length}\left(X\right),\mathrm{length}\left(Y\right)\right)} \tag{4}$$

例えば，生成コードが図 2(a) で正解コードが図 2(b) のときの STS は次のように計算する．まず，X に対する Y の編集距離を考える．X の左から `b` を `a` に，`j` を `i` に，`a` を `b` に，`i` を `j` にそれぞれ置換し，`+` と `1` を挿入すれば Y になる．よって，編集距離 $\mathrm{EditDistance}(X,Y) = 6$ となる．また，$\mathrm{length}(Y) = 13$，$\mathrm{length}(X) = 11$ なので

$$\begin{aligned}\mathrm{STS}(X,Y) &= 1 - 6/\max(13,11)\\ &\simeq 0.538\end{aligned}$$

となる．

本研究では，正解コードが複数ある場合の STS は式 (5) で算出する．

$$\mathrm{STS}(X,\,Ys) = \max_{Y \in Ys} \mathrm{STS}(X,Y) \tag{5}$$

ROUGE-L

ROUGE（Recall-Oriented Understudy for Gisting Evaluation） [15] は，自然言語要約の品質評価のために考えられた評価指標である．ROUGE-L [15] は ROUGE の一種で，2 つの列における最長共通部分列（Longest Common Subsequence: LCS）の長さを用いて計算される．ROUGE-L は，以下の 2 つの特徴を持つ．

- LCS を利用している
- F 値を利用している

ROUGE-L は，重み β を用いて式 (6) で表される．

$$\begin{aligned}
\text{ROUGE-L}(X,Y) &= \frac{(1+\beta^2)R_{\text{ROUGE-L}}(X,Y)P_{\text{ROUGE-L}}(X,Y)}{R_{\text{ROUGE-L}}(X,Y)+\beta^2 P_{\text{ROUGE-L}}(X,Y)} \\
R_{\text{ROUGE-L}}(X,Y) &= \frac{\text{LCS}(X,Y)}{\text{length}(Y)} \\
P_{\text{ROUGE-L}}(X,Y) &= \frac{\text{LCS}(X,Y)}{\text{length}(X)}
\end{aligned} \tag{6}$$

本研究では，$R_{\text{ROUGE-L}}$ と $P_{\text{ROUGE-L}}$ を同じ重要度として扱うために $\beta = 1$ とする．

例えば，生成コードが図 2(a) で正解コードが図 2(b) のときの ROUGE-L は次のように計算する．まず，X に対する Y の LCS を考える．LCS は “`s += [ ] + [ ]`” なので，$\text{LCS}(X,Y) = 7$ となる．また，$\text{length}(Y) = 13$，$\text{length}(X) = 11$ なので

$$\begin{aligned}
R_{\text{ROUGE-L}}(X,Y) &= 7/13 \\
P_{\text{ROUGE-L}}(X,Y) &= 7/11 \\
\text{ROUGE-L}(X,Y) &= \frac{2 \cdot 7/13 \cdot 7/11}{7/13 + 7/11} \\
&\simeq 0.614
\end{aligned}$$

となる．

本研究では，正解コードが複数ある場合の ROUGE-L は式 (7) で算出する．

$$\begin{aligned}
R_{\text{ROUGE-L}}(X,\, Ys) &= \frac{\sum_{Y \in Ys} \text{LCS}(X,Y)}{\sum_{Y \in Ys} \text{length}(Y)} \\
P_{\text{ROUGE-L}}(X,\, Ys) &= \frac{\sum_{Y \in Ys} \text{LCS}(X,Y)}{\text{length}(X)} \\
\text{ROUGE-L}(X,\, Ys) &= \frac{(1+\beta^2)R_{\text{ROUGE-L}}(X,\, Ys)P_{\text{ROUGE-L}}(X,\, Ys)}{R_{\text{ROUGE-L}}(X,\, Ys)+\beta^2 P_{\text{ROUGE-L}}(X,\, Ys)}
\end{aligned} \tag{7}$$

METEOR

METEOR は，自然言語の機械翻訳結果の評価のために考えられた評価指標で，BLEU の改良版として設計されている [16]．METEOR は，以下の 3 つの特徴を持つ．

- 完全一致以外のトークンの一致度を考慮している
- 全体的なトークンの並びを考慮している
- F 値を利用している

METEOR では，トークンの一致度を以下のように分類する．

(1)　完全一致

(2)　語幹が一致

(3)　同義語

(4)　フレーズが言い換え可能

ある文 A に含まれる単語 a と別の文 B に含まれる単語 b が一致するとき，一致度がどれに分類されるか判定する関数を $M(a,b)$ とする．このとき，A に含まれる単語であって B にも含まれる単語との一致度が（1）から（4）に分類される単語数をそれぞれ $m_1(A,B)$ から $m_4(A,B)$ で表す．特に，B が自明な場合は $m_i(A)$ と書く．また，A と B で同一順序で出現し，連続して一致する単語の列をチャンクという．チャンクの数を $ch(A,B)$ で表す．A と B のトークンの並びが近いほどチャンクの数は小さくなり，A と B が同じ文の場合チャンクの数は 1 となる．

METEOR は，生成されたテキスト内の内容語 h_c と 機能語 h_f，参照テキスト内の内容語 r_c と機能語 r_f，生成されたテキストと参照テキストそれぞれの一致する単語数の平均 m および単語の一致度ごとの重み w_i を用いて式 (8) で表される．

$$
\begin{aligned}
\mathrm{METEOR} &= (1-\mathrm{Pen})\cdot F_{\mathrm{METEOR}} \qquad (8)\\
\mathrm{Pen} &= \gamma\cdot\left(\frac{ch}{m}\right)^{\beta}\\
F_{\mathrm{METEOR}} &= \frac{P_{\mathrm{METEOR}}\cdot R_{\mathrm{METEOR}}}{\alpha\cdot P_{\mathrm{METEOR}}+(1-\alpha)\cdot R_{\mathrm{METEOR}}}\\
P_{\mathrm{METEOR}} &= \frac{\sum_i w_i\cdot(\delta\cdot m_i(h_c)+(1-\delta)\cdot m_i(h_f))}{\delta\cdot|h_c|+(1-\delta)\cdot|h_f|}\\
R_{\mathrm{METEOR}} &= \frac{\sum_i w_i\cdot(\delta\cdot m_i(r_c)+(1-\delta)\cdot m_i(r_f))}{\delta\cdot|r_c|+(1-\delta)\cdot|r_f|}
\end{aligned}
$$

α，β，γ，δ および w_i はパラメーターである．本研究では，ソースコードが英単語を用いて書かれる場合が多い点に着目して，生成コードと正解コードをそれぞれ生成されたテキストと参照テキストと見なして評価した．評価では，METEOR の実装である `meteor-1.5.jar` [17] に `-lang en` オプションを渡して言語設定を英語にし，コードを英語のテキストとみなした．各パラメーターは，言語設定を英語にしたときのデフォルトの値である．

本研究では，正解コードが複数ある場合の METEOR は式 (9) で算出する．

$$
\mathrm{METEOR}(X, Ys) = \max_{Y\in Ys}\mathrm{METEOR}(X,Y) \qquad (9)
$$

chrF

chrF（character n-gram F-score） [9] は，自然言語機械翻訳のために考えられた評価指標である．chrF は，以下の 2 つの特徴を持つ．

- 文字単位の n-gram を利用している

- F 値を利用している

chrF は，重み β を用いて式 (10) で表される．

$$\begin{aligned}
\mathrm{chrF}(X,Y) &= \frac{(1+\beta^2)R_{\mathrm{chrF}}(X,Y)P_{\mathrm{chrF}}(X,Y)}{R_{\mathrm{chrF}}(X,Y)+\beta^2 P_{\mathrm{chrF}}(X,Y)} \qquad (10)\\
R_{\mathrm{chrF}}(X,Y) &= \frac{1}{N}\sum_{n=1}^{N}\frac{\mathrm{length}\left(n\text{−grams}(X)\cap n\text{−grams}(Y)\right)}{\mathrm{length}\left(n\text{−grams}(Y)\right)}\\
P_{\mathrm{chrF}}(X,Y) &= \frac{1}{N}\sum_{n=1}^{N}\frac{\mathrm{length}\left(n\text{−grams}(X)\cap n\text{−grams}(Y)\right)}{\mathrm{length}\left(n\text{−grams}(X)\right)}
\end{aligned}$$

本研究では，$N=6$，$beta=3$ とする．これは，chrF を提案した研究 [9] で推奨された設定である．また，Python コードではインデントによってコードブロックを表現するため，本研究では chrF においてタブ文字も 1 文字として数えるものとする．

例えば，生成コードが図 2(a) で正解コードが図 2(b) のときの chrF は次のように計算する．まず，X に対する Y の文字単位の n-gram の適合率 $P_{\mathrm{chrF}}(X,Y)$ を考える．$n=1$ のとき，X の 1-gram は 12 個あり，そのうち Y にも存在する 1-gram は 12 個ある．同様に $n=2$ から 6 の場合も考えると

$$P_{\mathrm{chrF}}(X,Y)=\frac{1}{6}\left(\frac{12}{12}+\frac{8}{11}+\frac{4}{10}+\frac{1}{9}+\frac{0}{8}+\frac{0}{7}\right)$$

となる．また

$$R_{\mathrm{chrF}}(X,Y)=\frac{1}{6}\left(\frac{12}{14}+\frac{8}{13}+\frac{4}{12}+\frac{1}{11}+\frac{0}{10}+\frac{0}{9}\right)$$

となる．よって

$$\begin{aligned}
\mathrm{chrF}(X,Y) &= \frac{(1+3^2)R_{\mathrm{chrF}}P_{\mathrm{chrF}}}{R_{\mathrm{chrF}}+3^2P_{\mathrm{chrF}}}\\
&\simeq 0.321
\end{aligned}$$

となる．

本研究では，正解コードが複数ある場合の chrF は式 (11) で算出する．

$$\mathrm{chrF}(X,\,Ys)=\max_{Y\in Ys}\mathrm{chrF}(X,Y) \qquad (11)$$

提案指標

5 つの既存評価指標の各特徴に対してパラメーター a_i （$i = 1 \sim 15$）で重み付けしたものの和 f を提案指標として定義する（式 (12)）.

$$
\begin{aligned}
f(X,Y) = {} & a_1 \cdot \min\left(1, \exp\left(1 - a_2 \cdot \mathrm{length}(Y)/\,\mathrm{length}(X)\right)\right) \cdot \exp\left(\sum_{n=1}^{4} a_{n+2} \cdot \log p_n\right) \\
& + a_7 \cdot \mathrm{STS}(Y,X) \\
& + a_8 \cdot \frac{(1+a_9^2) R_{\text{ROUGE-L}}(X,Y) P_{\text{ROUGE-L}}(X,Y)}{R_{\text{ROUGE-L}}(X,Y) + a_9^2 P_{\text{ROUGE-L}}(X,Y)} \\
& + a_{10} \cdot \left(1 - a_{11} \cdot \left(\frac{ch}{m}\right)^{a_{12}}\right) \cdot \frac{P_{\text{METEOR}} \cdot R_{\text{METEOR}}}{a_{13} \cdot P_{\text{METEOR}} + (1-a_{13}) \cdot R_{\text{METEOR}}} \\
& + a_{14} \cdot \frac{(1+a_{15}^2) R_{\text{chrF}}(X,Y) P_{\text{chrF}}(X,Y)}{R_{\text{chrF}}(X,Y) + a_{15}^2 P_{\text{chrF}}(X,Y)}
\end{aligned}
\tag{12}
$$

ただし，a_i はそれぞれ以下の条件を満たす.

- $i \neq 2, 7$ について $a_i \in [0, \infty)$
- $a_2 \in (-\infty, \infty)$
- $a_7 \in [0, 1]$
- $a_1 + a_3 + a_4 + a_6 + a_{10} = 1$

また，正解コードが複数ある場合の提案指標は式 (13) で算出する.

$$
f(X, Ys) = \max_{Y \in Ys} f(X,Y) \tag{13}
$$

4　実験

本研究では，生成コードの修正容易性を測る被験者実験を 2 回実施した．各被験者実験の特徴を表 2 に示す．

1 回目の被験者実験（以下，実験 1）では，コード生成モデルを用いて生成したコードを要求を満たすコードへ修正する際に要する時間を測定する．測定した修正時間と評価指標による生成コードの評価値の相関をとる．負の相関が強いほど，その評価指標は自然言語で記述された要求からコードを生成する手法の評価に適していると言える．負の相関が強いということは，修正容易性が高いほど評価が高くなる傾向が強いからである．

2 回目の被験者実験（以下，実験 2）では，モデルが異なる場合にも適した評価指標が変わらないかどうかを確かめるために，1 回目の被験者実験とは異なるモデルによる生成コードを用いて，同様の実験を行った．

さらに，提案指標のパラメーター a_i を定めるために，提案指標による評価値と実験 2 で得られた修正時間との相関がなるべく大きくなるような a_i を調査した（実験 3）．

4.1　コード生成モデルの作成

実験 1

実験 1 で用いるコード生成モデル作成のために，GitHub 上で公開されている自然言語翻訳用のネットワーク[*1]を利用した．

学習には，プログラミングコンテストの問題文と正解コード，テストケースの組からなるデータセット ReCa [5] を用いた．データセットには，5,149 問の問題文と 16,673 個の Python コードが含まれて

表 2　実験 1 および実験 2 の特徴

	実験 1	実験 2
コード生成モデル	Seq2Seq を用いた GAN から作成	AlphaCode（Transformer ベース）
被験者数	11 名	13 名
題材	プログラミングコンテストの問題	プログラミングコンテストの問題
問題数	10 問	36 問
目的	生成コードの修正容易性の測定	生成コードの修正容易性の測定
内容	生成コードの修正	解答コード作成および生成コードの修正

[*1] https://github.com/nazim1021/neural-machine-translation-using-gan

いる．300 問をテスト用，200 問を検証用，残りを訓練用として学習を行いコード生成モデルを作成した．

作成したモデルの入出力例を図 3(b)(c)(d) に示す．入力は，図 3(a) のような英語の問題文に対して小文字化，レンマ化，ストップワード除去などの前処理を施した図 3(b) のような文である．出力は，図 3(c) のような Python コードのトークン列である．図 3(c) のようなトークン列は，適当な処理によって自動的に図 3(d) のような Python コードに変換できる．以下では，図 3(d) のようなコード生成モデルの出力から得られるコードを生成コードと呼ぶ．図 3(d) は要求を満たさないコードである．例えば，図 3(g) のようなテストケースの入力が与えられた場合に，図 3(d) は入力の受け取りに失敗する．入力の 1 行目には整数が 1 つしかないが，図 3(d) の 1 行目では `n` と `k` の 2 つの整数を受け取ろうとしているからである．問題の正答コードのひとつであり，要求を満たす図 3(e) では，図 3(g) の入力を与えると期待される出力と同じ出力が得られる．

```
Suppose we have a sequence of non-negative integers, Namely a_1, a_2, ..., a_n. At each time we can choose one term a_i with 0 < i < n and we subtract 1 from both a_i and a_i+1. We wonder whether we can get a sequence of all zeros after several operations.
```

(a) 問題文

```
suppose we have sequence of non-negative integer, namely a_1, a_2,..., a_n. at each time we can choose one term a_i with 0 < i < n and we subtract 1 from both a_i and a_i+1. we wonder whether we can get sequence of all zero after several operation.
```

(b) 前処理された問題文

```
n , k = map ( int , input ( ) . split ( ) ) &n a = list ( map ( int , input ( ) . split ( ) ) ) &n a . sort ( ) &n ans = 0 &n for i in range ( k ) : &n &indent if a [ i ] == a [ i - 1 ] : &n &indent &indent ans += 1 &n &indent &dedent print ( ans ) &n
```

(c) コード生成モデルの出力

```
n , k = map ( int , input ( ) . split ( ) )
a = list ( map ( int , input ( ) . split ( ) ) )
a . sort ( )
ans = 0
for i in range ( k ) :
    if a [ i ] == a [ i - 1 ] :
        ans += 1
print ( ans )
```

(d) 生成コード

図 3　実験 1 のデータの例

```
Len = int(input())
List = list(map(int, input().split()))
assume = True
res = 0
for i in range(Len):
    res = List[i] - res
    if res < 0:
        assume = False
        break
if assume and res == 0:
    print("YES")
else:
    print("NO")
```

(e) 正解コード

```
n = int ( input ( ) )
a = list ( map ( int , input ( ) . split ( ) ) )

for i in range ( n-1 ) :
    a[i+1]-=a[i]
    a[i]=0


if(a[-1]==0):print("YES")
else:print("NO")
```

(f) 修正済コード

入力

```
2
2 2
```

期待される出力

```
YES
```

(g) テストケース

図 3　実験 1 のデータの例（続き）

実験 2

実験 2 では，AlphaCode [18] というモデルの公開されている生成コードのサンプル[*2]を利用した．AlphaCode は，Transformer [19] ベースのネットワークを使用して作成されている．

サンプルでは，以下のような情報が提供されている．

- モデルへの入力である問題文
- モデルの出力である生成コード
- 生成コードがテストに通過したかどうか
- 問題がどのプログラミングコンテストで使われた問題か

問題文と生成コード，正解コード，生成コードを修正したコードの例を図 4 に示す．

[*2] `https://alphacode.deepmind.com/`

```
# RATING: 1200
# TAGS: greedy
# LANGUAGE IS python3
# CORRECT SOLUTION
# A string s of length n, consisting of lowercase letters of the English
# alphabet, is given.
#
# You must choose some number k between 0 and n. Then, you select k characters
# of s and permute them however you want. In this process, the positions of the
# other n-k characters remain unchanged. You have to perform this operation
# exactly once.
#
# For example, if s="andrea", you can choose the k=4 characters "a_d_ea" and
# permute them into "d_e_aa" so that after the operation the string becomes
# "dneraa".
#
# Determine the minimum k so that it is possible to sort s alphabetically (that
# is, after the operation its characters appear in alphabetical order).
#
# Input
#
# The first line contains a single integer t (1 ≤ t ≤ 1000) — the number of test
# cases. Then t test cases follow.
#
# The first line of each test case contains one integer n (1 ≤ n ≤ 40) — the
# length of the string.
#
# The second line of each test case contains the string s. It is guaranteed that
# s contains only lowercase letters of the English alphabet.
#
# Output
#
# For each test case, output the minimum k that allows you to obtain a string
# sorted alphabetically, through the operation described above.
#
# Example
#
# Input
#
#
# 4
# 3
# lol
# 10
# codeforces
# 5
# aaaaa
# 4
# dcba
#
#
# Output
#
#
# 2
# 6
# 0
# 4
#
# Note
#
# In the first test case, we can choose the k=2 characters "_ol" and rearrange
# them as "_lo" (so the resulting string is "llo"). It is not possible to sort
# the string choosing strictly less than 2 characters.
#
# In the second test case, one possible way to sort s is to consider the k=6
# characters "_o__force_" and rearrange them as "_c__efoor_" (so the resulting
# string is "ccdeefoors"). One can show that it is not possible to sort the
# string choosing strictly less than 6 characters.
#
# In the third test case, string s is already sorted (so we can choose k=0
# characters).
#
# In the fourth test case, we can choose all k=4 characters "dcba" and reverse
# the whole string (so the resulting string is "abcd").
```

(a) 問題文

図 4　実験 2 のデータの例

```
t=int(input())
for i in range(t):
    n=int(input())
    s=input()
    k=0
    for j in range(n):
        if s:
            if s[:j+1]!=sorted(s[:j+1]) or s[:j+1]!=s[:j+1][::-1]:
                k+=1
                s=s[j+1:]+s[:j+1]
        else:
            break
    print(k)
```

(b) 生成コード

```
t = int(input())
while t:
    n = int(input())
    s = input()
    ss = sorted(s)

    x = 0
    for i in range(n):
        if s[i] != ss[i]:
            x += 1
    print(x)
    t -= 1
```

(c) 正解コード

```
t=int(input())
for i in range(t):
    n=int(input())
    s=input()
    k=0
    for j in range(n):
        if s:
            if s[j]!=sorted(s)[j]:
                k+=1
        else:
            break
    print(k)
```

(d) 修正済コード

図 4　実験 2 のデータの例（続き）

4.2 対象

実験 1

実験 1 には，テスト用データ 300 問からランダムにサンプリングした 10 問を用いた．サンプリングされた問題には，1 問あたり平均 53.4 個のテストケースがある．

被験者は，大阪大学大学院情報科学研究科に所属する教員 1 名，同研究科に所属する修士の学生 8 名，大阪大学基礎工学部情報科学科に所属する学部生 2 名の計 11 名である．被験者ごとに Python の習熟度は異なっていた．習熟していない被験者のために，コードの修正に必要と思われる処理のチートシートを配布した．

実験 2

実験 2 には，AlphaCode のサンプルとして提供されている全 38 問のうち，問題の内容が重複する 2 つの問題を除いた 36 問を用いた．修正済コードは，実際にコンテストサイトに投稿し，用意されたすべてのテストケースに通るかどうかを確かめた．

被験者は，大阪大学大学院情報科学研究科に所属する修士の学生 9 名，大阪大学基礎工学部情報科学科に所属する学部生 4 名の計 13 名である．各被験者の Python の習熟度を測るために，“Python の利用頻度は以下のうちどれになりますか？ ” という質問に以下の 1 から 4 の選択肢で答えてもらうアンケートを実施した．

1. 使ったことがない
2. 使ったことがある
3. たまに利用している
4. 日常的に利用している

アンケートの結果，3 の “たまに利用している” と回答したのが 8 名，2 の “使ったことがある” と回答したのが 5 名となった．

4.3 方法

実験 1

被験者実験は，問題ごとに以下の手順で行った．

STEP-1 （要求の理解） 被験者には図 3(a) のような問題文と図 3(g) のようなテストケースが与えられる．被験者は問題文を読み，要求を理解する．

STEP-2 （生成コードの修正） 被験者には図 3(d) のような生成コードが与えられる．被験者は STEP-1 で与えられた問題の解答コードとして正しいコードになるように生成コードを修正する．すなわち，要求を満たすように生成コードを修正する．

STEP-3 （テスト） STEP-2 で修正したコードが STEP-1 で与えられたすべてのテストケースに通過するか確かめる．通過すれば STEP-3 は終了する．通過しなければ STEP-2 に戻り生成コードをさらに修正する．

生成コードを修正するまでに要した時間として，STEP-2 から STEP-3 終了までの秒数を測定した．以上の手順で図 3(f) のように修正された生成コードを，修正済コードと呼ぶ．STEP-2 や STEP-3 において生成コードを修正できなかった場合は，修正時間は得られない．

実験 2

正解コードを得るために，被験者ごとに生成コードを用いずに問題へ解答する問題と生成コードの修正を行う問題を分けた．具体的にはまず，被験者と問題をそれぞれ 2 つのグループに分けた．被験者は，グループ間で Python の利用頻度に偏りが出ないように A グループと B グループに分けた．問題は，プログラミングコンテスト運営側が想定した難易度によってグループ間で偏りがでないように p グループと q グループに分けた．次に，A グループの被験者には q グループの問題の修正，B グループの被験者には p グループの問題の修正を実験 1 と同様の手順で実施してもらった．最後に，A グループの被験者には p グループの問題の解答，B グループの被験者には q グループの問題の解答を問題ごとに以下の手順で実施してもらった．

STEP-1 （要求の理解） 被験者には問題文が与えられる．被験者は問題文を読み，要求を理解する．

STEP-2 （解答コードの作成） 被験者は STEP-1 で与えられた問題の解答コードを作成する．修正の場合とは異なり，生成コードは与えられない．

STEP-3 （テスト） STEP-2 で作成したコードがプログラミングコンテストで用意されているすべてのテストケースに通過するか確かめる．通過すれば STEP-3 は終了する．通過しなければ STEP-2 に戻り解答コードを修正する．

解答コードを作成するまでに要した時間（以下，作成時間）として，STEP-2 から STEP-3 終了までの秒数を測定した．得られた解答コードは，生成コードを各評価指標で評価する際の正解コードとして用いた．

実験 3

提案指標の最適なパラメーター a_i を定めるために，提案指標による評価値と以下の 3 つの値と評価値の相関がなるべく大きくなるような a_i をそれぞれ求める実験を行った．

実験 3-1　実験 1 で得られた修正時間

実験 3-2　実験 2 で得られた修正時間

実験 3-3　実験 1 で得られた修正時間と実験 2 で得られた修正時間の和

そのような a_i は, 非線形計画問題を解くことによって求めることができる．本研究では, basin hopping 法と Sequential Least Squares Programming を用いて乱数のシードを変えながらそれぞれ 5 回解を求め，最良時のパラメーターを最適なパラメーターとみなした．また，それぞれの実験で得られたパラメーターの提案指標を実験 1 および実験 2 の生成コードに適用し，修正時間との相関を調査した．

4.4　結果と考察

実験 1

表 3 に各評価指標の評価値と修正時間のピアソンの相関係数および有意性検定の結果である p 値を示す．表 3 から，評価値と修正時間の相関は chrF が最も強いことがわかる．また，METEOR と chrF に関しては，有意水準 5 % で有意に相関があることもわかる．

実験 2

表 4 に各評価指標の評価値と修正時間のピアソンの相関係数および有意性検定の結果である p 値を示す．なお，修正時間が長すぎる結果を外れ値として除外するために，修正時間が 1 時間を超えるものは省いて相関を調査した．表 4 から，評価値と修正時間の相関は BLEU が最も強いことがわかる．ただし，どの評価指標も有意水準 5 % で有意に相関があるとは言えないこともわかる．

表 3　実験 1 における各評価指標の評価値と修正時間の相関

評価指標	相関係数	p 値
chrF	−0.316	0.00536
METEOR	−0.251	0.0286
BLEU	−0.181	0.117
STS	−0.100	0.389
ROUGE-L	0.0115	0.921

実験 3

実験 3-1 で得られた提案指標の最適なパラメーターを表 5 に示す．表 5 を見ると，BLEU に対して約 81 %，METEOR に対して約 19 % の重みがかかっており，その他の評価指標は最終的な評価値にほとんど寄与していない．BLEU に関しては，n-gram の適合率にかかっている重みが 0 に近く，これも最終的な評価値にほとんど寄与していない．つまり，BLEU にかかっている重みは生成コードの長さが正解コードの長さよりどれだけ長くてもよいかを表す部分にかかっている．METEOR に関しては，a_{12} が式 (8) の β よりも大きいものの，a_{11} が式 (8) の γ より小さいことから，トークンの並びが合っ

表 4　実験 2 における各評価指標の評価値と修正時間の相関

評価指標	相関係数	p 値
BLEU	−0.137	0.169
chrF	−0.125	0.209
METEOR	−0.117	0.242
STS	−0.101	0.312
ROUGE-L	−0.0964	0.335

表 5　実験 3-1 で得られた提案指標の最適なパラメーター

a_1	0.81
a_2	1.09
a_3	5.28×10^{-15}
a_4	1.81×10^{-16}
a_5	1.14×10^{-16}
a_6	2.32×10^{-16}
a_7	0.0
a_8	9.12×10^{-17}
a_9	1.37
a_{10}	0.190
a_{11}	0.265
a_{12}	1.08
a_{13}	0.0
a_{14}	0.0
a_{15}	1.42

ているかどうかを重視する度合いはそれほど変わっていない．また，a_{13} が 0.0 となっており，トークンの 1-gram の適合率のみを評価に用いるようになっている．

パラメーターを表 5 に設定した提案指標を用いた実験 1 と実験 2 それぞれの生成コードの評価値と修正時間の相関を表 6 に示す．表 6 に示した相関係数を表 3 や表 4 と比べると，最適化した実験 1 の相関係数は既存の評価指標を上回っている一方で，実験 2 では ROUGE-L を除く 4 つの既存評価指標よりも悪い結果となっている．

実験 3-2 で得られた提案指標の最適なパラメーターを表 7 に示す．表 7 を見ると，BLEU に対して約 71 %，ROUGE-L に対して約 23 %，METEOR に対して約 5 % の重みがかかっており，STS と chrF は最終的な評価値にほとんど寄与していない．BLEU に関しては，a_3，a_5，a_6 がほぼ 0 であり a_4 だけがほぼ 1 となっているので，ほとんどトークン単位の 2-gram だけが評価に用いられている．ROUGE-L に関しては，a_9 がほぼ 0 なのでほとんど LCS の適合率で評価している．METEOR に関

表 6　パラメーターを表 5 にした提案指標の評価値と修正時間の相関

	相関係数	p 値
実験 1	-0.415	1.91×10^{-4}
実験 2	-0.0632	0.528

表 7　実験 3-2 で得られた提案指標の最適なパラメーター

a_1	0.713
a_2	1.1
a_3	1.63×10^{-18}
a_4	0.977
a_5	3.20×10^{-19}
a_6	0.0
a_7	0.0
a_8	0.232
a_9	6.94×10^{-8}
a_{10}	0.0549
a_{11}	2.75×10^{-18}
a_{12}	1.27
a_{13}	1.61
a_{14}	2.28×10^{-18}
a_{15}	2.94

しては，a_{11} がほぼ 0 なのでトークンの並びはほとんど考慮しておらず，1-gram の再現率を重視した F 値で評価している．

パラメーターを表 7 に設定した提案指標を用いた実験 1 と実験 2 それぞれの生成コードの評価値と修正時間の相関を表 8 に示す．表 8 を見ると，実験 3-1 と同様に，最適化した実験 2 の相関係数は既存の評価指標を上回っている一方で，実験 1 では chrF や METEOR よりも悪い結果となっている．

実験 3-3 で得られた提案指標の最適なパラメーターを表 9 に示す．表 9 を見ると，BLEU に対して約 97 %，METEOR に対して約 3 % の重みがかかっており，他の既存の評価指標は最終的な評価値にほとんど寄与していない．BLEU に関しては，a_3 から a_6 のうち a_4 以外はほぼ 0 となっている．a_4 についても約 0.07 と小さいことから，トークン単位の 2-gram がわずかに用いられているだけで，全体としては生成コードと正解コードの長さでほとんど評価値が決まっている．METEOR に関しては，a_{12} が 0 かつ a_{11} が約 0.74 と高く METEOR 自体にかかっている重み a_{10} も小さいことから，最終的な評

表 8　パラメーターを表 7 にした提案指標の評価値と修正時間の相関

	相関係数	p 値
実験 1	-0.201	8.17×10^{-2}
実験 2	-0.476	4.15×10^{-7}

表 9　実験 3-3 で得られた提案指標の最適なパラメーター

a_1	0.971
a_2	0.860
a_3	0.0
a_4	0.0738
a_5	0.0
a_6	0.000415
a_7	0.0
a_8	0.0
a_9	1.88
a_{10}	0.0282
a_{11}	0.738
a_{12}	0.0
a_{13}	1.61
a_{14}	0.000777
a_{15}	16.8

価値にはほとんど影響していないと考えられる．

パラメーターを表 9 に設定した提案指標を用いた実験 1 と実験 2 それぞれの生成コードの評価値と修正時間の相関を表 10 に示す．表 10 を見ると，実験 1 と実験 2 の両方で既存の評価指標による相関を上回っていることが分かる．

RQ1: 修正時間を用いて測定する生成コードの修正容易性と関連の強い評価指標はどれか？

表 3 を見ると相関が最も強い評価指標は chrF となっている．つまり，実験 1 で使用したモデルの生成コードの修正容易性と最も強い関連を持つ評価指標が chrF であると言える．よって，対象とした既存の評価指標の中では chrF が実験 1 のモデルに対して最も適切である．

表 4 を見ると相関が最も強い評価指標は BLEU であり，次点で chrF となっている．つまり，実験 2 で使用したモデルの生成コードの修正容易性と最も強い関連を持つ評価指標が BLEU であり，次点で chrF が強い．ただし，それらは有意水準 5 % で有意に相関があるとは言えず，たまたま関連を持つという結果が得られたにすぎない可能性がある．

以上の結果を総合すると，既存の評価指標の中では chrF が生成コードの評価に最も適切であると考えられる．ただし，その評価値と修正時間の相関は弱い．

RQ1 への回答： 生成コードの修正時間と関連の強い評価指標は chrF であり，chrF は本研究で対象とした評価指標の中では生成コードの評価に比較的適していた．しかし，その相関は弱く，実験 1 で用いたモデルの生成コードに対する相関係数は -0.316，実験 2 では -0.125 であった．

RQ2: 生成コードの評価に適した評価指標はモデルごとに異なるか？

表 3 と表 4 を相関係数の強さ順に着目して比べると，BLEU の順位が 3 位と 1 位で大きく異なっている．

表 5 と表 7 を比べると，実験 1 では生成コードと正解コードの長さの比とトークン単位の 1-gram の適合率を重視する評価指標が適している一方で，実験 2 ではトークン単位の 2-gram の適合率と LCS の F 値を重視する評価指標が適しているとわかる．また，表 6 では実験 1 に最適化した評価指標が実験 2 では低い相関を示していることがわかる．表 8 でも同様に，実験 2 に最適化した評価指標が実験 1

表 10　パラメーターを表 9 にした提案指標の評価値と修正時間の相関

	相関係数	p 値
実験 1	-0.367	1.1×10^{-3}
実験 2	-0.467	7.46×10^{-7}

では低い相関を示していることがわかる．

以上から，生成コードの評価に適した評価指標はモデルごとに異なると考えられる．

RQ2 への回答： 生成コードの評価に適した評価指標はモデルごとに異なっていた．既存の評価指標では，実験 1 で用いたモデルの生成コードの評価には chrF が，実験 2 で用いたモデルの生成コードの評価には BLEU が適していた．また，実験 1 で用いたモデルの生成コードに適した評価指標はコードの長さやトークン単位の 1-gram に着目した評価指標であり，実験 2 で用いたモデルの生成コードに適した評価指標はトークン単位の 2-gram と LCS を着目した評価指標であった．

RQ3: どのような特徴を持つ評価指標が生成コードの評価に適しているか？

実験 3-1 および実験 3-2 から，実験 1 と実験 2 で用いたそれぞれのモデルの生成コードに適した評価指標が持つ特徴は RQ2 への回答で示した．

どちらのモデルが生成したコードの評価にも適しており，モデル間での比較を可能とする評価指標が持つ特徴を実験 3-3 の結果から考察する．表 9 をみると，生成コードと正解コードのトークン列の長さの比に着目した評価指標が生成コードの評価に適した評価指標ということになる．

しかし，このような評価指標が真に生成コードの評価に適した評価指標だとは考えられない．例えば，図 5 のようなコードについて考える．図 5(a) はバブルソートの実装コード，図 5(a) は FizzBuzz の実装コードの一部，図 5(c) は FizzBuzz の完全な実装コードである．図 5(a) と図 5(c) は機能的に異なっている．また，関数になっている点やループが何重になっているかという点などで構造的にも異なっている．さらに，`return`や `for`など片方にのみ登場するトークンも多く存在している．このように 2 つのコードは全く異なるが，図 5(a) を生成コード，図 5(c) を正解コードとしてパラメーターが表 9 の提案指標で評価すると 0.829 という評価値が得られる．図 5(b) を生成コード，図 5(c) を正解コードとしてパラメーターが表 9 の提案指標で評価すると 0.0440 という評価値が得られる．要求として FizzBuzz 問題が与えられたとき，バブルソートのコードが渡された場合と書きかけの FizzBuzz のコードが渡された場合では後者の方が修正が容易だと考えられるが，提案指標による評価では前者の方が評価が高くなる．一方で，パラメーターが表 7 の提案指標で評価すると順に 0.145 と 0.249 という評価値が得られる．この場合，後者の方が評価が高くなっており，この例においては妥当な評価ができていると考えられる．

既存の評価指標を用いた評価では，パラメーターが表 9 のような妥当とは考えられない提案指標よりも低い相関しか得られなかった．これは既存の評価指標が持つ特徴は生成コードの評価観点として不十分だからである可能性が考えられる．一方で，RQ2 への回答で示したように，トークン単位の 1-gram や 2-gram，LCS などの既存の評価指標の特徴を組み合わせることである程度よい評価指標は作れる可

```
n = int(input())
a = list(map(int, input().split()))
for i in range(len(a)):
    for j in range(len(a) - 1):
        if a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]
print("␣".join(map(str, a)))
```

(a) バブルソートの実装コード

```
def fizzbuzz(n):
    return str(n)
```

(b) FizzBuzz の実装コードの一部

```
def fizzbuzz(n):
    if n % 3 == 0 and n % 5 == 0:
        return "FizzBuzz"
    elif n % 3 == 0:
        return "Fizz"
    elif n % 5 == 0:
        return "Buzz"
    else:
        return str(n)
```

(c) FizzBuzz の完全な実装コード

図 5　パラメーターが表 9 の場合の提案指標が評価に適さない例

能性がある．

RQ3 への回答： トークン単位の 1-gram や 2-gram，LCS を組み合わせて用いる評価指標は生成コードの評価に適する可能性がある．しかし，そのような既存の評価指標が持つ特徴の組み合わせでは不十分な可能性があり，さらなる研究が必要である．

5　妥当性の脅威

実験は，実験 1 では 11 人の被験者と最大 10 問分のデータ，実験 2 では 13 人の被験者と最大 36 問分のデータを用いて実施された．これらは統計的な分析をするには規模が小さすぎる可能性がある．より多くの被験者とデータ数で実験した場合，同様の結果が得られるとは限らない．

実験の被験者は実験 1，実験 2 ともに大阪大学に所属するメンバーであり，所属や環境，開発経験，Python への習熟度などが異なる被験者も含む場合と比べてコーディングやコードリーディング，問題の理解の際にバイアスがかかっている可能性がある．例えば，大阪大学基礎工学部情報科学科で必修となっている講義で類似の問題を解いたことがあるような場合には，特定の問題だけ修正時間が早くなることが考えらえる．

実験 1 および実験 2 ではプログラミングコンテストの問題を対象としている．他のドメインを対象とした場合には異なる実験結果が得られる可能性がある．

RQ2 において，モデル間の比較が可能な評価指標の調査のために 2 種類のモデルの結果を用いた．他のコード生成モデルを利用すると適した評価指標は変わるかもしれない．

6 関連研究

6.1 自然言語処理分野における評価指標の調査

自然言語の文章が出力されるタスクを対象として，文章の評価に適した評価指標を明らかにする研究はある [20, 21]．例えば，自然言語翻訳 [22] や自然言語要約 [23] のための研究がある．これらの研究では，各タスクに適している評価指標について調査している．本研究はコードの評価に適した評価指標について調査しているという点で，これらの研究とは異なっている．

6.2 コード生成以外の分野における評価指標の調査

本研究に近い分野の研究として，コードマイグレーションタスクに適した評価指標に関する研究がある [6]．コードマイグレーションは，ある言語で書かれたコードを入力として，別の言語で書かれたコードを出力するタスクである．彼らは，tree edit distance や graph edit distance を用いた評価指標と BLEU を対象として比較する調査を行っている．本研究は，2 つの点でこの研究とは異なっている．1 つ目は，構文誤りを含むコードに対しても適用可能な評価指標を対象にしている点である．tree edit distance や graph edit distance は構文誤りを含むコードに対しては適用できない．本研究では，STS や BLEU だけでなく，ROUGE-L や METEOR，chrF を対象として調査した結果，生成コードの評価には chrF がよいことを明らかにした．2 つ目は，コード生成タスクを対象にしている点である．彼らは，コードマイグレーションタスクを対象として，生成コードと正解コードの意味的な類似度を測定する被験者実験を行っている．本研究では，コード生成技術の使われ方に注目し，生成コードの修正容易性を測定する被験者実験を行った．コード生成に限らず，コードマイグレーションなどについても，修正容易性による評価の方が実用的な可能性がある．

6.3 コード生成分野における評価指標の調査

本研究室と同じコード生成分野における評価指標の研究として，生成コードと正解コードの類似度の人による評価と各評価指標の関連を調査した研究がある [24]．本研究は，評価指標が人のどういった評価と関連を持つべきと考えているかという点でこの研究とは異なっている．彼らは，コードの類似度を人が 5 段階で評価した結果と各評価指標による評価値との関連を調査している．本研究では，開発者が生成コードを修正する容易性と評価値との関連を調査している．

7　おわりに

本研究では，構文誤りを含む自動生成コードに対しても適用可能で評価に適した指標を明らかにすることを目的に，被験者実験によって得られた生成コードの修正容易性と関連の強い評価指標を調査した．調査の結果，生成コードの修正時間と評価指標による評価値の相関が最も強く，生成コードの評価に比較的適した指標は chrF であったが，相関係数は最大で -0.316 と小さかった．また，生成コードの評価に適した指標はモデルごとに異なるという結果も得られた．さらに，評価指標が持つ特徴を調査した結果，トークン単位の 1-gram や 2-gram，LCS などを組み合わせて用いる評価指標は生成コードの評価に適することがわかった．ただし，本研究で対象とした自然言語処理用の評価指標が持つ特徴の組み合わせではモデル間の比較にも使用可能な評価指標は作成できない可能性があり，今後さらなる評価指標の研究が必要である．

今後の研究課題として，既存の評価指標が持っていない特徴の導入を考えている．例えば，構文誤りを含むコードにおいても部分的な抽象構文木が得られることに着目し，その部分木を用いることで構造的な特徴を組み込める可能性がある．また，コード生成モデルの性能と生成コードの評価に適した評価指標が持つ特徴の関係の調査も考えている．本研究で用いたモデルよりもさらに高性能なモデルにおいては，n が 3 以上の n-gram を用いる評価指標が適している可能性がある．加えて，プログラミングコンテスト Codeforces の問題に対してだけでなく異なるドメインに対しても実験を行い，本研究の一般性を示すことも今後の課題である．

謝辞

本研究を行うにあたり多くの皆様に多大な支援をいただきました．

楠本真二教授．本研究を行うにあたり，丁寧でわかりやすい指導をしていただきました．また，あたたかな励ましと差し入れは研究の活力となりました．

肥後芳樹教授．担当教員として 2 年間，研究におけるあらゆる場面で助言と励ましをいただきました．研究の方向性で迷ったときも実験が上手くいかなかったときも多くの時間を費やして一緒に考え，研究を導いていただきました．論文執筆時の添削や研究発表の練習にもたくさんの時間を割いていただき，数多くの貴重な意見をくださりました．

柗本真佑助教．本研究の方向性や改善案について有益な助言を多数いただきました．また，研究発表における改善に関しても多数の指摘をいただきました．

事務補佐員の橋本美砂子さん．研究活動にあたって必要な様々な手続きをしていただき，研究に集中できる環境を作っていただきました．特に，本研究の実験参加者への手続きに関して大変お世話になりました．研究生活においても数多くの励ましの言葉と支援をいただきました．

倉林利行さんをはじめとする共同研究先である日本電信電話株式会社ソフトウェアイノベーションセンタの方々．毎月のミーティングにおいて多くの助言と多様な意見をいただきました．

同じ研究室の皆様．研究に関しては，相談を含め多様な物事について意見を交わしていただき多くの学びを得られ，豊かで快適な研究室生活をおくれました．特に，同期の皆様には研究生活における様々な場面で手助けや励ましをいただきました．また，本研究で実施した実験への協力をいただけたことで，本研究を進めることができました．

両親．これまで様々な面で応援と支援をしてもらいました．とても感謝しています．

参考文献

[1] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. GraphCodeBERT: Pre-training code representations with data flow. In *Proc. International Conference on Learning Representations*, 2021.

[2] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai Wei Chang. Unified pre-training for program understanding and generation. In *Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, June 2021.

[3] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2017.

[4] Li Dong and Mirella Lapata. Coarse-to-Fine decoding for neural semantic parsing. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2018.

[5] Hui Liu, Mingzhu Shen, Jiaqi Zhu, Nan Niu, Ge Li, and Lu Zhang. Deep learning based program generation from requirements text: Are we there yet? *IEEE Transactions on Software Engineering*, Vol. 48, No. 4, 2022.

[6] Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. Does BLEU score work for code migration? In *Proc. IEEE/ACM International Conference on Program Comprehension*, 2019.

[7] Gang Zhao and Jeff Huang. Deepsim: deep learning code functional similarity. In *Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018.

[8] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[9] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proc. Workshop on Statistical Machine Translation*, 2015.

[10] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. `https://arxiv.org/abs/1704.07535`, 2017.

[11] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *Proc. International Conference on*

Learning Representations, 2017.

[12] Lee Spector. Autoconstructive evolution: Push, PushGP, and Pushpop. In *Proc. Genetic and Evolutionary Computation Conference*, 2001.

[13] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2002.

[14] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10, 1966.

[15] Chin Yew Lin. ROUGE: a package for automatic evaluation of summaries. In *Proc. ACL Workshop on Text Summarization Branches Out*, 2004.

[16] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proc. ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

[17] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proc. Workshop on Statistical Machine Translation*, 2014.

[18] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, CyprienMasson d' Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, Vol. 378, No. 6624, 2022.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems*, 2017.

[20] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. Evaluation of text generation: A survey. `https://arxiv.org/abs/2006.14799`, 2020.

[21] Nitika Mathur, Timothy Baldwin, and Trevor Cohn. Tangled up in BLEU: Reevaluating the evaluation of automatic machine translation evaluation metrics. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2020.

[22] Shweta Chauhan and Philemon Daniel. A comprehensive survey on various fully automatic machine translation evaluation metrics. *Neural Processing Letters*, 2022.

[23] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang,

and Hongbin Sun. On the evaluation of neural code summarization. In *Proc. International Conference on Software Engineering*, 2022.

[24] Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. Out of the BLEU: how should we assess quality of the Code Generation models? `https://arxiv.org/abs/2208.03133`, 2022.