修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学研究科　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	稲村 健太郎	学籍番号	2431027
論 文 題 目	難易度とブルーム・タキソノミーのマルチタスク強化学習を組み込んだプログラミング問題の自動生成		

要　旨

プログラミング教育の重要性が高まる中，CBT を用いた大規模なプログラミングテストの実施が進み，アイテムバンク構築に向けて多数の問題を準備する必要がある．しかし，作問には大きな労力とコストがかかる．また，CBT では項目反応理論（IRT）に基づく連続値の難易度が用いられるため，IRT の難易度を指定して問題を生成できることが望ましい．この課題に対し，大規模言語モデル（LLM）を用いて難易度を指定して問題生成を行う手法が提案されている．一方，教育評価の観点では，改訂版ブルーム・タキソノミーに基づく細目標のように評価目標を明確にした問題生成も重要である．しかし，細目標をプロンプトで指定する既存手法はLLMを細目標に整合するよう直接最適化していないため，生成される問題が指示した細目標を満たす保証がない．そこで本研究では，Codeforces のプログラミング問題を対象に，プロンプトで指定した難易度と細目標（Remember/Understand/Apply）の双方を満たす問題（問題文および正解コード）を生成する強化学習手法を提案する．提案手法では，細目標ラベルのみを与えるのではなく，各細目標に対して要求される思考過程や許容される解法の性質を具体的に記述したプロンプトを導入し，細目標に関する制約を明示した問題生成を行う．ここで，LLM の最適化に必要な報酬を計算するためには，生成された問題から難易度と細目標を予測する必要がある．そこで本研究では，生成問題から抽出した特徴量を入力として，難易度予測と細目標分類を同時に遂行する Multi-gate Mixture-of-Experts（MMoE）を導入する．MMoE はタスクごとのゲート機構により，タスク間で共有すべき表現とタスク固有表現を分離でき，難易度予測と細目標分類の負の転移を抑制しつつ両タスクの予測精度を向上できる．次に，指定難易度と予測難易度の差に基づく難易度報酬と，指定細目標の予測確率に基づく細目標報酬の重み付き和からなる報酬関数を設計し，方策勾配法により問題生成LLMを最適化する．評価実験の結果，MMoEはシングルタスクモデルと比較して予測精度が向上することを示した．さらに，細目標報酬と細目標の定義を具体化したプロンプトの双方を導入した提案手法は，難易度のみを最適化する手法，細目標報酬を導入して難易度と細目標の双方を最適化する手法，および細目標の定義を具体化したプロンプトを導入した上で難易度のみを最適化する手法と比較して，難易度誤差を同程度に保ちつつ細目標の達成度を向上できることを示した．また，正解コードの分析より，指定難易度が高くなるにつれてコード行数が増加する傾向，および指定細目標が Remember, Understand, Apply と高くなるにつれてコード行数や認知的複雑度が高くなる傾向を確認した．さらに，生成問題の追従性を検証するため，GPT-4o-mini の API およびエキスパート評価により難易度（5 段階）と細目標（3 段階）を判定した結果，指定難易度・指定細目標に対して一定の追従性が確認された．評価結果の比較より，全体としてはエキスパート評価の方が指定条件との整合をより強く反映する傾向が見られた．

2025 年度 情報数理工学 (MI) プログラム
修士論文

難易度とブルーム・タキソノミーのマルチタスク強化学習を組み込んだプログラミング問題の自動生成

2026 年 3 月 12 日

電気通信大学 情報数理工学プログラム
学籍番号 2431027

稲村 健太郎

主任指導教員 植野 真臣

副指導教員 宇都 雅輝

目次

1　まえがき

近年, 将来の情報技術者不足への対応のため, プログラミング教育の重要性はますます高まっている [1, 2]. プログラミング能力評価のため, CBT（Computer Based Testing）を活用した大規模なプログラミングテストが実施されつつあり [3], アイテムバンク構築に向けて多数の問題を準備する必要がある. しかし, プログラミング問題の作成には多大な労力とコストがかかる課題がある.

初期の問題生成手法として, 大規模言語モデル（LLM）を用いて教育目的の問題やパズルを自動生成する手法が提案されてきた [4, 5, 6, 7, 8]. しかし, これらの既存手法はいずれも生成される問題やパズルの難易度を調整できなかった. CBT では, 難易度パラメータを持つ項目反応理論（Item Response Theory; IRT）を用いることが一般的であり, 作問者は所望の難易度を想定して問題を作成する. そのため, LLM による問題自動生成でも IRT における難易度を指定できることが望ましく, 近年, 難易度調整可能な問題生成の研究が多くなされている [9, 10, 11, 12, 13, 14, 15, 16, 17, 18].

Nguyen ら（2023）は, 3 段階の難易度を LLM へのプロンプト中で指定し, 学習者の技能水準に応じたプログラミング演習問題を自動生成する問題生成手法を提案している [9]. Li ら（2024）は, 「答えが 1 文で足りるか・複数文にまたがるか」と「答えが文中に明示されているか・推論が必要か」で難易度をラベル化し質問文を生成する手法を提案した [10]. さらに, Li ら（2025）は, 3 段階の難易度（easy・medium・hard）をプロンプト中で指定して質問文を生成する手法を提案した [11]. また, Wu ら（2025）は 3 段階の難易度（easy・medium・hard）をプロンプト中で指定し, 質問文の作成計画を生成させてから質問文を生成する手法を提案した [12]. しかし, これらはいずれも難易度を離散カテゴリとして扱うため, CBT でよく用いられる IRT の難易度には対応できない. そのため, 難易度を連続値として入力して, 所望の問題を自動生成する手法も提案されている [13, 14, 15, 16, 17, 18].

Pourcel ら（2024）は, プログラミング問題回答 LLM の正解率から算出される連続値の難易度をプロンプト中で指定するとともに指定難易度に近い問題をプロンプトに入力し, Python プログラミング問題を生成する手法を提案した [13]. しかし, 本手法は問題生成 LLM を目標難易度に合わせて学習・制御しないため, 生成される問題が指示した範囲の難易度を満たす保証がない. さらに, 学習者の能力と問題の難易度の関係性を考慮していないため, 個々の学習者の能力に合わせた難易度を指定して問題を生成できない.

これらの問題点を改善するために, Tomikawa ら（2024, 2025）は, IRT を用いて難易度調整が可能な読解問題生成手法を提案した [14, 15, 16]. まず Tomikawa ら（2024）は, 読解対象文および

指定難易度を含むプロンプトを入力とし, 指定難易度に近い問題文・正答を出力するように LLM を SFT 学習することで, 指定難易度に応じた問題生成を実現している [14, 15]. しかし, この SFT 学習では訓練データ中の問題の単語列に対する尤度を最大化するように LLM を訓練しており, 難易度調整の精度を目的関数として最適化していないため, 難易度調整精度に改善の余地が残る.

そこで Tomikawa ら（2025）は, 読解対象文および指定難易度を含むプロンプトを入力とし, 指定難易度に近い問題文・正答を「望ましい出力（preferred）」, 遠いものを「望ましくない出力（rejected）」として構成したデータを用いて, DPO 学習により難易度調整精度を目的関数として LLM を直接最適化している [16].

また, Sarkar ら（2025）は, 問題文の言い換えによって難易度予測が変動し得る点に着目し, 問題文を入力として難易度を予測する BERT モデルの頑健性・予測精度を向上させる手法を提案している [17]. 同手法では, 受検者の正解率に基づいて元の問題文に付与された難易度と同程度の予測難易度となる言い換え文が生成されるよう, その差が小さいほど高い報酬を与え, 方策勾配法 [8, 17] により LLM を最適化した.

教育評価の観点では, 評価目標を明確にすることが重要である [19]. 例えば, 同じ難易度の問題でも, 改訂版ブルーム・タキソノミー [19] の細目標において暗記（Remember）を問う問題と応用（Apply）を問う問題では測定する能力が異なり得る. すなわち, 指定した難易度に近い問題が生成できても, 目標とする能力を測定する問題になっているとは限らない. そのため, 生成される問題が難易度だけでなく評価目標に整合するように, 問題生成 LLM を直接最適化することが望ましい. Chen ら（2025）は改訂版ブルーム・タキソノミー [19] に基づく細目標を入力プロンプトで指定して多肢選択式問題を生成する手法（KAQG）を提案している [18]. しかし, 同手法は問題生成 LLM を細目標に整合するように学習していないため, 生成される問題が指示した細目標を満たす保証がない.

また, 複数のタスクを同時に学習するマルチタスク学習を用いた質問生成手法も提案されている [20, 21]. Xia ら（2023）は, 長文を入力として質問を生成する際に重要なフレーズが選択されるように, 重要フレーズ選択と質問生成を同時学習する手法を提案している [20]. この手法では, 質問生成のクロスエントロピー損失とフレーズ選択のクロスエントロピー損失の重み付き和を最小化することで Transformer を最適化している. また, Ding ら（2024）は, 長文の段落および解答を入力として質問を生成する際に, 段落中の重要な文を手がかりとするように, 質問生成のクロスエントロピー損失と重要文同定のクロスエントロピー損失の和を最小化することで BART を最適化している [21].

本研究では Codeforces で出題されたプログラミング問題を対象とし, 難易度, および改訂版ブルーム・タキソノミーに基づく細目標をプロンプトで指定するとともに, 細目標ごとに要求される

思考過程や許容される解法の性質を具体的に記述したプロンプトを用いる．その上で, 生成された問題が両者に整合するほど高い報酬を与える強化学習により問題生成 LLM を最適化することで, 指定した難易度と細目標の双方を満たすプログラミング問題生成手法を提案する．各問題の難易度は受検者の反応データから IRT の難易度を推定し, 各問題が要求する細目標は, ブルーム・タキソノミーに基づき, Remember/Understand/Apply の 3 段階で専門家が付与する．

ここで, プログラミング問題を解くために要求される細目標は問題文・正解コードの構造や情報統合の度合いと関係し, 難易度と相関する．そのため, 難易度予測と細目標分類を同時に学習し, 問題文・正解コードに由来する共通の手がかりを活用しつつタスク固有の違いも考慮することで, 両タスクの予測精度向上が期待できる．そこで本研究では, 難易度予測と細目標の分類を同時に遂行する Multi-gate Mixture-of-Experts（MMoE）[22] を導入し, 生成された問題の難易度および細目標を学習・予測する．Codeforces データを用いた実験により, 難易度または細目標の一方のみを予測する場合と比較して予測精度が向上することを示す.

提案手法では, 指定難易度と予測難易度の差が小さいほど大きい値となる難易度報酬, および細目標の予測確率である細目標報酬の重み付き和に基づく報酬関数を提案し, 方策勾配法 [8, 17] により問題生成 LLM を最適化する．さらに, 細目標ごとに要求される思考過程や解法の性質を具体的に記述したプロンプトを導入することで, 生成時に細目標に関する制約を与える． 強化学習実験の結果, 細目標報酬と細目標の定義を具体化したプロンプトの双方を導入する提案手法は, 難易度のみを最適化する手法, 細目標報酬を導入して難易度と細目標の双方を最適化する手法, および細目標の定義を具体化したプロンプトを導入した上で難易度のみを最適化する手法と比較して, 難易度調整精度を保ったまま, 細目標の達成度を向上できることを示す.

2　従来の難易度調整可能な問題生成手法

難易度調整可能な問題生成手法として, これまでに以下のような手法が提案されている.

2.1　3 段階の難易度を指定して問題生成を行う手法

Nguyen ら（2023）は, 学習者ごとにカスタマイズされたプログラミング演習問題を自動生成する手法を提案した [9]. 従来の手法では, 個々のスキルレベルに応じた問題作成の負担が大きく, 特に初学者向けの問題が不足していた．また, 生成された問題の品質や実行可能性が保証されないという課題もあった．これらの課題を解決するために, 彼らは LLM に対して「easy」「intermediate」「hard」の 3 段階の難易度と, プログラミング概念（例: string）を指定してプログラミング問題生成を行わせた．生成には, Zero-shot プロンプトや, 既存の演習問題を例示する Few-shot プロンプ

トが用いられた．さらに, 生成された問題に対してコンパイルチェックや LLM が生成した単体テストの実行, LLM による難易度分類による自動フィルタリングを行い, 指定した難易度に合致し, かつ, 実行可能な問題のみを選別した．しかし, 本手法における難易度指定は 3 段階の離散値に限られるため, 粒度が粗い．そのため, 学習者の能力に合わせた細やかな難易度調整は不可能である．

2.2 Few-shot プロンプトとアーカイブ活用型自動問題生成

生成後の問題の難易度を離散値ではなく連続値で評価する自動問題生成手法として, Pourcel ら（2024）は, 多様なスキルと高い難易度を持つプログラミング問題を生成する手法「Autotelic CodE Search（ACES）」を提案した [13]．本手法では, 難易度を $d = 100(1 - \text{pass@1})$ と定義し, 「アーカイブ」と呼ばれるデータベースに, スキルや難易度情報が付与された既存問題を蓄積した．ここで pass@1 は LLM ソルバーによる $N = 50$ 回試行での正解率であり, pass@1 $= 0$ の問題は棄却される．新たな問題を生成する際は, 目標とするスキル構成に近く, かつ高難易度の問題をアーカイブから選出した．これらを Few-shot の例としてプロンプトに組み込むことで, LLM に高難易度な問題の生成を促した．生成された問題は, 同様に LLM ソルバーを用いて難易度 d が評価され, LLM による自動ラベリングで判定されたスキルラベルとともにアーカイブに追加された．アーカイブに追加された問題は, 問題生成の際に LLM に入力されるプロンプト文に組み込まれる Few-shot 例として活用されることで, 多様なスキルセットと高い難易度を両立した効率的な自動問題生成を目指した．しかし, このアプローチには以下の課題が存在する．第一に, 生成される問題の質が Few-shot として与えられる既存問題の分布に依存する．そのため, モデル自体が目標難易度に合わせて最適化されるわけではなく, 指定した難易度の問題が確実に生成される保証がない．第二に, 難易度の定義が LLM ソルバーの正答率に基づいている．これは学習者の能力と問題の難易度の関係性を考慮しておらず, 実際の受験者にとっての難易度を反映していない．第三に, 生成モデル自体を学習させる仕組みがないため, 所望の難易度の問題を生成する能力が獲得されない．

2.3 仮想学習者データと DPO を用いた難易度調整

学習者の能力に応じた難易度の問題を生成できる手法として, Tomikawa ら（2025）は, 項目反応理論（IRT）[23] と Direct Preference Optimization（DPO）を用いた多枝選択式読解問題の生成手法を提案した [16]．本手法では, 自己回帰型モデル（Llama 2）を用い, 指定された難易度に対応した問題を生成することを目指した．しかし, モデルの学習に用いるデータセットには難易度が付与されていなかった．そこで彼らは, 複数の QA システムを仮想的な学習者集団とみなし, こ

れらの QA システムの正誤反応データから IRT を用いて難易度を推定し, 学習データセットに付与した. その後, 難易度が付与された学習データセットで自己回帰型モデルの教師あり学習を行った. さらに, 指定難易度に近い問題を生成するように, モデルを DPO で最適化した. RACE データセットを用いた実験では, DPO による追加訓練を行ったモデルが, 教師あり学習のみのモデルや Few-shot プロンプト法よりも高い難易度調整精度を示した. しかし, 難易度の推定が仮想的な学習者（QA システム）に基づいている点に課題がある. QA システムの能力分布は, 実際の学習者集団の能力分布と一致するとは限らない. したがって, 指定した難易度が, 実際の学習者にとって適切な難易度であるという保証はない.

2.4 実受験者データに基づく難易度予測と報酬モデリング

実際の学習者集団に適合した難易度に合致する問題文の生成手法として, Sarkar ら（2025）は, 実受験者の正解率に基づく難易度を真値とした難易度予測機の最適化手法を提案した [17]. 彼らは, 従来の難易度予測手法において, 問題の予測難易度が「表現の言い回し」に左右される点に着目し, 表現が変化しても予測値が一貫するよう, 予測機の頑健性を高めることを目指した. 具体的には, まず実際の受験者の正解率から算出された難易度を「真値」とし, これを予測する BERT ベースのモデルを学習させた. 次に, この難易度予測機を報酬モデルとして固定し, 元の問題文と意味が保たれるパラフレーズを生成するように LLM を強化学習させた. ここでは, LLM が生成したパラフレーズの予測難易度が真値に近いほど高い報酬を与えることで, 難易度制御機能を有したパラフレーズ生成を実現した. その後, 生成されたデータを用いて予測機を再学習させることで, BEA 2024 データセットにおいて従来法比 12.46 %の RMSE 低減を達成した. この手法に含まれる LLM の強化学習フレームワークは, 実データに基づく報酬モデルを用いて生成モデルを最適化するという点で, 難易度調整機能を有する問題生成に有効である. しかし, この手法をプログラミング問題生成に応用する場合, 以下の課題が残る. 第一に, 最適化の対象が「難易度」という単一の指標に限定されている点である. 教育においては, 問題の難易度（解きにくさ）だけでなく, その問題が学習者のどのような細目標（記憶, 理解, 応用など）を要求するかを考慮する必要がある [19]. しかし, 本手法ではこの細目標を制御できないため, 学習目標に合致した問題生成が保証されない. 第二に, 複数の指標を同時に扱う際のモデル構造の課題である. 指標ごとに独立した予測機を用意すれば, 指標間の有益な相関（正の転移）を活用できない. 一方で, 単純に一つのモデルでパラメータを共有させると, タスク間の干渉（負の転移）が生じ, 予測精度が低下する恐れがある [22]. したがって, 難易度と細目標の双方を高精度に推定し, 安定した報酬を与えるためには, タスク間の関係性を考慮して負の転移を抑制しつつ, 有益な情報を共有できる, より高度な報酬モ

デルのアーキテクチャが必要となる.

3 提案手法

本研究では, 指定された出題難易度, およびブルーム・タキソノミー [19] に基づく細目標の双方に合致したプログラミング問題を自動生成する手法を提案する. 提案手法では, 生成される問題の難易度と細目標を同時に最適化することで, 単なる解きづらさの調整だけでなく, 意図する細目標に沿った問題生成を可能にする. さらに, 細目標ラベルだけでなく, 各細目標に対して要求される思考過程や許容される解法の性質を具体的に記述したプロンプトを生成時に入力することで, 細目標に関する制約の下で問題生成を行う. 提案手法では, このようなプロンプト上の制約を与えた上で, 生成された問題（問題文・正解コード）が指定難易度 d および指定細目標 l にどの程度合致しているかを定量化し, その値を報酬として LLM を強化学習により最適化する.

この報酬を計算するためには, 生成された問題から難易度と細目標を予測する必要がある. そこで本研究では, 生成された問題から抽出した特徴量を入力として, 難易度予測と細目標分類を同時に行う予測モデルとして Multi-gate Mixture-of-Experts（MMoE）[22] を導入する. MMoE はタスクごとのゲート機構により, 共有すべき表現とタスク固有表現を柔軟に切り分ける構造を持つ. この構造により, 難易度予測と細目標分類という性質の異なるタスク間において負の転移を抑制しつつ有益な情報を共有することが可能となるため, 従来の単一指標モデルや単純なマルチタスクモデルでは困難であった, 双方の条件に適した高精度な報酬信号の生成を実現する. 以降では, 提案手法の全体フローを示した後, プロンプトに基づく問題生成, 難易度・細目標の予測, 予測結果に基づく報酬の計算, ならびに強化学習による最適化手順を詳細に述べる.

3.1 提案手法の概要

図 1 に提案手法の全体フローを示す. 提案手法では, まず, 指定難易度 d, 指定細目標 $l \in \{\text{Remember}, \text{Understand}, \text{Apply}\}$, 出題領域, および各細目標に対して要求される思考過程や許容される解法の性質を含むプログラミング問題生成プロンプトを作成する（図 1①）. 次に, このプログラミング問題生成プロンプトを LLM（Llama-3.2-3B-Instruct）に入力し, 問題文と正解コードの最尤系列 $\hat{Y}$, サンプリング系列 Y^s をそれぞれ生成させる（図 1②）. 最尤系列 $\hat{Y}$ およびサンプリング系列 Y^s から, BERT 系モデル（CodeBERT）により問題文と正解コードの埋め込みベクトルを抽出し, 問題文の埋め込みベクトルと正解コードの埋め込みベクトルのコサイン類似度を算出する. その後, 問題文の埋め込みベクトル, 正解コードの埋め込みベクトル, および両者のコサイン類似度を結合し, 特徴量ベクトル $\hat{\boldsymbol{x}}$, $\boldsymbol{x}^s$ とする（図 1③）. 作成された特徴量ベクトル $\hat{\boldsymbol{x}}$,

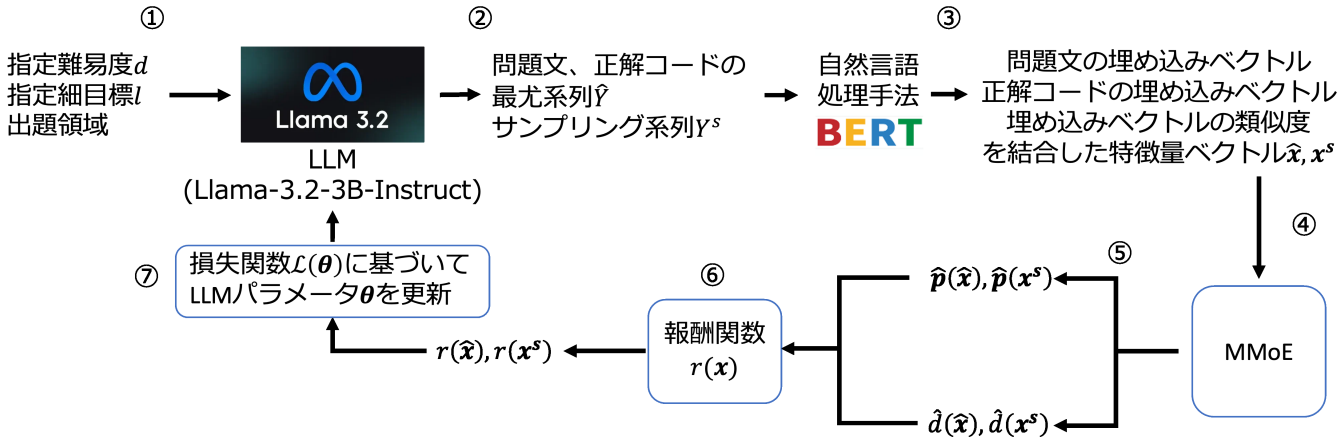

図 1: 提案手法の全体フロー

$\boldsymbol{x}^s$ を MMoE に入力し（図 1④），予測難易度 $\hat{d}(\hat{\boldsymbol{x}}), \hat{d}(\boldsymbol{x}^s)$ と細目標の各クラス確率 $\hat{\boldsymbol{p}}(\hat{\boldsymbol{x}}), \hat{\boldsymbol{p}}(\boldsymbol{x}^s)$ を出力する（図 1⑤）．これらをもとに，報酬 $r(\hat{\boldsymbol{x}}), r(\boldsymbol{x}^s)$ を計算する（図 1⑥）．最後に，損失関数 $\mathcal{L}(\boldsymbol{\theta})$ を計算し，方策勾配法 [8, 17] により LLM のパラメータ $\boldsymbol{\theta}$ を最適化する（図 1⑦）．これらの手順を繰り返し，指定難易度 d と指定細目標 l を同時に満たす問題が生成されやすくなるように LLM のパラメータ $\boldsymbol{\theta}$ を更新する．

3.2 LLM による問題生成

指定難易度 d と指定細目標 $l \in \{\text{Remember}, \text{Understand}, \text{Apply}\}$，および出題領域タグを含むプロンプト X を作成する．プロンプト X として，以下の 2 種類のいずれかを用いる．1 つ目のプロンプトは，指定細目標ごとにコード行数の制約を入力する．具体的には，指定細目標が Remember の場合はコード行数を 5 行以内にする指示をプロンプトに入力する．また，指定細目標が Understand または Apply の場合はコード行数を 5 行以上にする指示をプロンプトに入力する（以降，「難易度・細目標指定プロンプト 1」とする）．2 つ目のプロンプトは，コード行数の制約に加えて，指定細目標の詳細な定義をプロンプトに入力する．具体的には，指定細目標ごとに以下の文章をプロンプトに入力する（以降，「難易度・細目標指定プロンプト 2」とする）．

■指定細目標が Remember の場合

```
- The task must be a direct implementation of an explicitly described procedure.
- Use at most ONE control structure (‘for‘/‘while‘/‘if‘) in the entire solution:
either a single loop (‘for‘/‘while‘), OR a single ‘if‘ (no nesting).
```

- If a loop (‘for‘/‘while‘) is used, do not use ‘if‘ anywhere in the correct answer code. If an ‘if‘ is used, do not use any loop.
- Do not require algorithmic design or multi-step reasoning. Any necessary rule/formula must be explicitly stated in the question text.

■指定細目標が Understand **の場合**

- The main difficulty must be interpreting the statement and translating it into a clear answer form (equation/conditions/procedure).
- Allowed standard techniques/data structures (use at most ONE of them as the main idea; do not require strategy selection by tight constraints): sorting, set/dict for duplicate check or frequency counting, two-pointers/sliding window, straightforward BFS/DFS, template-form simple DP/greedy, light bitmask enumeration, binary search (monotonicity is explicit; no need to discover it), Dijkstra using heapq.
- Allowed reformulation of the question text (at most ONE):
  (1) After sorting, solve by a simple one-pass/adjacent-check.
  (2) Turn the task into duplicate/frequency counting.
  (3) Reduce to checking a condition for an integer T where, as T increases (or decreases), the answer changes only in ONE direction (from Yes to No, or from No to Yes), and this one-direction change is explicitly stated or immediately clear from the statement.

■指定細目標が Apply **の場合**

- The task must require algorithmic design based on constraints/efficiency; it must NOT be solvable by direct translation alone.
- The constraints must rule out at least one naive approach, forcing an efficiency-based design choice (e.g., switching from brute force to DP/ binary search/graph methods).
- Require at least ONE reasoning-driven core idea:
  (1) Find the minimum/maximum feasible integer T using a Yes/No feasibility check; feasibility changes only in ONE direction as T increases (or decreases),

```
and this one-direction property is not explicitly given and must be identified
and used in the search.
  (2) Use an operation-invariant property/value to decide the answer.
  (3) Construct and output a valid object/sequence by a designed procedure.
  (4) Solve an optimization problem with a greedy choice justified by an
exchange/swap idea: replacing part of any solution with the greedy choice does
not worsen feasibility/optimality (no written proof required).
  (5) Use DP that requires designing the state and transitions; do not use a
pre-given DP formula.
- Allowed standard techniques/data structures: sorting, set/dict for duplicate
check or frequency counting, two-pointers/sliding window, BFS/DFS, DP/greedy,
bitmask enumeration, threshold search (binary search) with a Yes/No feasibility
check (monotonicity not explicit), Dijkstra using heapq.
- It is allowed (and encouraged) to combine up to TWO of the allowed standard
techniques/data structures.
```

上記の難易度・細目標指定プロンプト 1 または難易度・細目標指定プロンプト 2 のいずれかを X として，LLM $P_{\boldsymbol{\theta}}$ に X を入力する（図 1①）．そして, プログラミング問題の問題文と正解コードの最尤系列 $\hat{Y}$, サンプリング系列 Y^s をそれぞれ生成させる [8]（図 1②）．

$$\hat{Y} = \underset{Y}{\operatorname{argmax}} P_{\boldsymbol{\theta}}(Y \mid X)$$
$$Y^s \sim P_{\boldsymbol{\theta}}(Y \mid X)$$

3.3 生成された問題の特徴量抽出

MMoE に入力して難易度と細目標を予測するために, 生成された問題（問題文と正解コード）$\hat{Y}$, Y^s からそれぞれ特徴量ベクトル $\hat{\boldsymbol{x}}$, $\boldsymbol{x}^s$ を抽出する（図 1③）．まず, 問題文および正解コードの各テキストをトークナイズし, BERT 系モデル（CodeBERT）に入力して最終層の隠れ状態を得る．本研究では, 隠れ状態からパディングを除外したマスクド平均（mean pooling）により固定長ベクトルを算出し, L2 正規化を行い, 埋め込みベクトルとする．次に, 問題文の埋め込みベクトルと正解コードの埋め込みベクトルのコサイン類似度を算出する．最後に, 問題文の埋め込みベクトル, 正解コードの埋め込みベクトル, および両者のコサイン類似度を連結し, 特徴量ベクトルとする．

3.4 MMoE による細目標・難易度の予測

本節では, 生成された問題（問題文・正解コード）から難易度と細目標を予測するモデルについて述べる. 提案手法の強化学習では, 生成物が指定難易度 d および指定細目標 l にどの程度合致しているかを報酬として与える必要がある. しかし, 生成された問題に対して難易度・細目標の正解ラベルを直接観測することはできないため, 生成された問題からこれらを予測するモデルを用いて報酬を計算する.

また, 細目標（Remember/Understand/Apply）は問題文・正解コードの構造や情報統合の度合いと関係し, 難易度とも一定の相関をもつ. そこで, 難易度予測と細目標分類を同時に学習し, 共通の手がかりを活用しつつタスク固有の差異も考慮できる Multi-gate Mixture-of-Experts（MMoE）[22] を難易度・細目標の予測モデルとして導入する. 以降では, 提案手法で用いる MMoE の入出力と構造, および学習方法を述べる.

3.4.1 MMoE の入出力

MMoE は, 3.3 節で定義した特徴量ベクトル $\boldsymbol{x} \in \mathbb{R}^d$ を入力とし, 予測難易度 $\hat{d}(\boldsymbol{x})$ と, 細目標のクラス確率 $\hat{\boldsymbol{p}}(\boldsymbol{x}) = [\hat{p}_{\text{Remember}}(\boldsymbol{x}), \hat{p}_{\text{Understand}}(\boldsymbol{x}), \hat{p}_{\text{Apply}}(\boldsymbol{x})]$ を同時に出力する.

LLM が生成した最尤系列 $\hat{Y}$ およびサンプリング系列 Y^s から得た特徴量ベクトル $\hat{\boldsymbol{x}}$, $\boldsymbol{x}^s$ を MMoE に入力し（図 1④）, $(\hat{d}(\hat{\boldsymbol{x}}), \hat{\boldsymbol{p}}(\hat{\boldsymbol{x}}))$, $(\hat{d}(\boldsymbol{x}^s), \hat{\boldsymbol{p}}(\boldsymbol{x}^s))$ を得る（図 1⑤）.

3.4.2 MMoE の構造

図 2 に本研究で用いる MMoE の構造を示す. 本研究で用いる MMoE のタスク数は $T = 2$ であり, $t = 1$ を難易度予測, $t = 2$ を細目標分類に対応付ける. MMoE は, 複数のエキスパートネットワーク $\{f_e(\cdot)\}_{e=1}^{E}$ と, タスク固有のゲーティングネットワーク $\{g^t(\cdot)\}$, およびタワーネットワークの 3 つの主要なネットワークにより構成される. 各エキスパートネットワークは, 入力 $\boldsymbol{x}$ から, タスク間で共通する特徴を学習する. 次に, 各タスク固有のゲーティングネットワークは, エキスパートに異なる重みを割り当てることで, 各タスクに対する共通特徴の重要度を学習する. これらの重み付けされた特徴量は, 対応するタワーネットワークへの入力として用いられ, 予測のためのタスク固有の特徴を学習する. MMoE は, タスク間で共有される特徴表現を学習するために, E 個のエキスパートネットワーク $(e = 1, 2, \cdots, E)$ で構成される. 各エキスパートネットワークは, R 個の隠れ層 $(r = 1, 2, \cdots, R)$ を持つ多層フィードフォワードニューラルネットワークとして実装される. e 番目のエキスパートネットワークにおいて, r 番目の隠れ層の出力 $\boldsymbol{L}_r^e$ は, 線形変換と

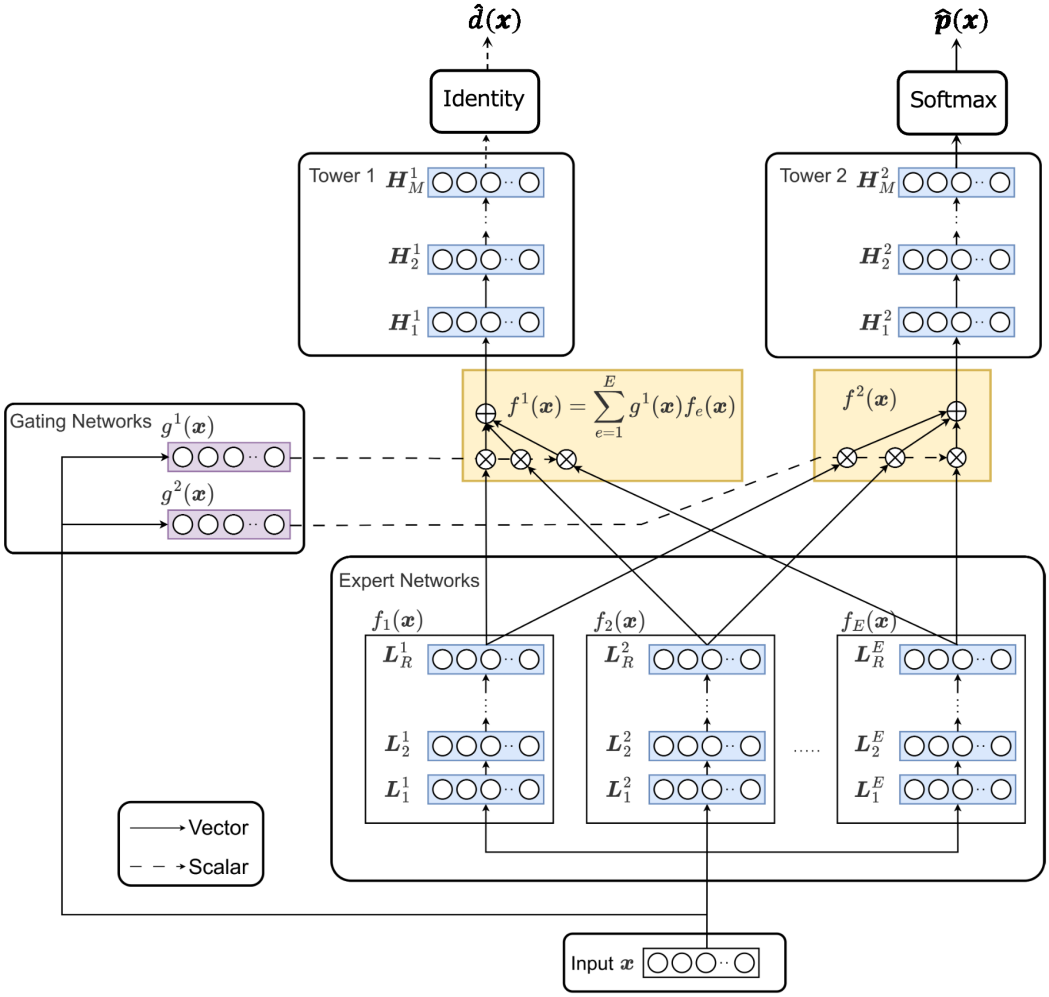

図 2: MMoE の構造

ReLU 活性化関数を用いて以下のように計算される.

$$\boldsymbol{L}_r^e = \begin{cases} \mathrm{ReLU}(\boldsymbol{W}_r^e \boldsymbol{x} + \boldsymbol{b}_r^e), & r = 1, \\ \mathrm{ReLU}(\boldsymbol{W}_r^e \boldsymbol{L}_{r-1}^e + \boldsymbol{b}_r^e), & r = 2, \cdots, R, \end{cases}$$

ここで, $\boldsymbol{W}_r^e$ と $\boldsymbol{b}_r^e$ はそれぞれ, e 番目のエキスパートネットワークにおける r 番目の隠れ層の重み行列とバイアスベクトルを表す. さらに, 各エキスパートから出力される特徴表現は以下の通りとなる.

$$\boldsymbol{f}_e(\boldsymbol{x}) = \boldsymbol{L}_R^e.$$

次に, 各タスク $t \in \{1,2\}$（難易度予測/細目標分類）に対して, タスク固有のゲーティングネットワークは, 各エキスパートの特徴表現 $\boldsymbol{f}_e(\boldsymbol{x})$ の重要度を学習する.

$$\boldsymbol{g}^t(\boldsymbol{x}) = \mathrm{softmax}(\boldsymbol{W}^t \boldsymbol{x})$$

$$\boldsymbol{f}^t(\boldsymbol{x}) = \sum_{e=1}^{E} g_e^t(\boldsymbol{x})\, \boldsymbol{f}_e(\boldsymbol{x}),$$

ここで, $\boldsymbol{W}^t \in \mathbb{R}^{E \times d}$ はタスク t 固有の重み行列を表す. また, $\boldsymbol{g}^t(\boldsymbol{x}) = [g_1^t(\boldsymbol{x}), \ldots, g_E^t(\boldsymbol{x})]$ はエキスパートに対する重みを表す.

次に, ゲーティングネットワークから得られたタスク固有の特徴表現 $\boldsymbol{f}^t(\boldsymbol{x})$ は, M 個の隠れ層 $(m = 1, 2, \cdots, M)$ からなるタスク固有のタワーネットワークに入力される. タスク t のタワーネットワークにおいて, m 番目の隠れ層の出力 $\boldsymbol{H}_m^t$ は, 線形変換と ReLU 活性化関数を用いて以下のように計算される.

$$\boldsymbol{H}_m^t = \begin{cases} \mathrm{ReLU}(\boldsymbol{W}_m^t \boldsymbol{f}^t(\boldsymbol{x}) + \boldsymbol{b}_m^t), & m = 1, \\ \mathrm{ReLU}(\boldsymbol{W}_m^t \boldsymbol{H}_{m-1}^t + \boldsymbol{b}_m^t), & m = 2, \cdots, M, \end{cases}$$

ここで, $\boldsymbol{W}_m^t$ と $\boldsymbol{b}_m^t$ はそれぞれ, タスク t の m 番目の隠れ層の重み行列とバイアスベクトルを表す.

最後に, タスク固有の予測 $\hat{y}^t$ は, 最終層の出力 $\boldsymbol{H}_M^t$ にタスク依存の出力活性化関数 $\psi^t(\cdot)$ を適用して得られる.

$$\hat{y}^t = \psi^t(\boldsymbol{H}_M^t).$$

ここで, $\psi^t(\cdot)$ はタスク t の予測目的に対応する出力活性化関数である. 本研究では, タスク $t = 1$（難易度予測）に対して $\hat{y}^1 = \hat{d}(\boldsymbol{x})$ を出力し（ψ^1 は恒等関数）, タスク $t = 2$（細目標分類）に対して $\hat{y}^2 = \hat{\boldsymbol{p}}(\boldsymbol{x})$ を出力する（ψ^2 は softmax 関数）.

以上により, MMoE は入力 $\boldsymbol{x}$ に応じてエキスパートの寄与をタスクごとに調整し, 共通表現の共有とタスク固有表現の分離を両立することで, 難易度予測と細目標分類の双方を高精度に行う.

3.4.3 MMoE の学習

MMoE の学習は, 4.1 節で構築する Codeforces データセットを用いた教師あり学習によって行う. 損失関数 $\mathcal{L}_{\mathrm{MMoE}}$ は, 難易度予測の平均二乗誤差（MSE）と, 細目標分類の交差エントロピー誤差の加重和（CE）として以下のように定義し, これを最小化する.

$$\mathcal{L}_{\mathrm{MMoE}} = \lambda \cdot \mathrm{MSE}(d, \hat{d}) + (1 - \lambda) \cdot \mathrm{CE}(l, \hat{p}) \tag{1}$$

ここで, d は正解難易度, $\hat{d}$ は予測難易度, l は正解ラベル, $\hat{p}$ は予測確率分布である. λ は損失の重み係数である.

3.5 報酬の計算

予測難易度 $\hat{d}(\boldsymbol{x})$ および指定細目標 l の予測確率 $\hat{p}_l(\boldsymbol{x})$ を用いて, 報酬 $r(\boldsymbol{x})$ を以下で定義する.

$$r(\boldsymbol{x}) = \mu \cdot r_{\text{diff}}(\boldsymbol{x}) + (1-\mu) \cdot r_{\text{bloom}}(\boldsymbol{x}) \tag{2}$$

ここで, $\mu \in [0,1]$ は両報酬の重み係数である. $r_{\text{diff}}(\boldsymbol{x})$ は指定難易度 d への近さに基づく難易度報酬, $r_{\text{bloom}}(\boldsymbol{x})$ は指定細目標 l への合致度に基づく細目標報酬であり, それぞれ次式で与える.

$$r_{\text{diff}}(\boldsymbol{x}) = \exp\left(-\frac{(\hat{d}(\boldsymbol{x}) - d)^2}{2\sigma^2}\right) \tag{3}$$

$$r_{\text{bloom}}(\boldsymbol{x}) = \hat{p}_l(\boldsymbol{x}) \tag{4}$$

ここで, $\sigma > 0$ は難易度報酬の許容幅を制御するパラメータであり, 式 3 はガウスカーネルにより, 生成された問題の予測難易度と指定難易度の差を報酬に変換するものである. また, $r_{\text{diff}}(\boldsymbol{x}) \in (0,1]$ および $r_{\text{bloom}}(\boldsymbol{x}) \in [0,1]$ より, μ により両報酬の相対的な寄与を調整できる. 式 2 を用いて, 最尤系列に対する報酬 $r(\hat{\boldsymbol{x}})$ とサンプリング系列に対する報酬 $r(\boldsymbol{x}^s)$ をそれぞれ算出する（図 1⑥）.

3.6 生成モデルの最適化

生成モデル（LLM）に対し, 指定した条件を満たす問題を生成させるために, 学習済みの MMoE を用いた強化学習を行う. 本研究では, 生成の安定性を高めるため, 最尤推定による生成結果（ベースライン）とサンプリングによる生成結果を比較して学習する手法を採用する [8]. 生成モデルの最適化には, 期待報酬を最大化する強化学習を用いる. 勾配の分散を低減させるため, 最尤系列 $\hat{Y}$ の報酬 $r(\hat{\boldsymbol{x}})$ をベースラインとして利用した強化学習アルゴリズム [8] を用いる. 入力プロンプトを X とし, サンプリング系列 $Y^s = (y_1^s, \ldots, y_{|Y^s|}^s)$ に対して, 以下の損失関数 $\mathcal{L}(\boldsymbol{\theta})$ を定義し, これを最小化することで LLM のパラメータ $\boldsymbol{\theta}$ を更新する（図 1⑦）. ただし, 以下のいずれかに当てはまる場合は LLM のパラメータ $\boldsymbol{\theta}$ を更新しない.

- サンプリング系列 Y^s がプロンプトで指示した出力形式を満たさない場合
- サンプリング系列 Y^s の正解コードが実行不可能である場合
- サンプリング系列 Y^s の正解コードの行数がプロンプト中の指示を満たさない場合

$$\mathcal{L}(\boldsymbol{\theta}) = \big(r(\hat{\boldsymbol{x}}) - r(\boldsymbol{x}^s)\big) \sum_{\tau=1}^{|Y^s|} \log P_{\boldsymbol{\theta}}(y_\tau^s \mid X, y_{<\tau}^s)\,. \tag{5}$$

上式において, $r(\boldsymbol{x}^s)$ がベースライン報酬 $r(\hat{\boldsymbol{x}})$ よりも高い場合（すなわち, $r(\hat{\boldsymbol{x}}) - r(\boldsymbol{x}^s) < 0$）, 損失が負の値となり, 最小化によってサンプリング系列 Y^s の生成確率 $P_{\boldsymbol{\theta}}(Y^s)$ が増加する. 逆にサンプリング系列がベースラインより劣る場合は, その生成確率は抑制される. これにより, 生成モデルはより高い報酬が得られる系列を生成するように最適化される.

4 評価実験

4.1 学習データセットの構築

本研究では, MMoE の学習および生成モデル（LLM）の Supervised Fine-Tuning (SFT) 学習, および, Direct Preference Optimization (DPO) 学習に用いるデータセットとして, プログラミングコンテストサイト Codeforces から収集したアルゴリズム問題を使用した. データセットの構築は以下の手順で行った. まず, 出題領域として, Kim ら [24] の手法に準拠し, Codeforces において最も頻出する上位 5 種類のアルゴリズムタグ（implementation, data structures, greedy, math, dp）を選定した. なお, 1 つの問題に複数のタグが付与されている場合があり, タグ情報が欠落している問題は除外した. 次に, 各問題文に対応する正解コードを統合した. 具体的には, Codeforces の提出データ [25] から正解（Accepted）と判定された Python コードを抽出し, 対応する問題文と紐付けた. 正解コードが存在しない問題は除外した. さらに, コンテスト実施中の全ユーザーの回答ログを Codeforces API 経由で取得し, 項目反応理論（IRT）[23] を用いて各問題の難易度を推定し付与した. 回答ログが得られず難易度推定が困難な問題は, データセットから除外した. 加えて, 各問題が要求する細目標を定義するため, ブルーム・タキソノミーの改訂版 [19] に定義されている細目標のうち, Remember, Understand, Apply の 3 段階のラベルを用いる. 各問題の問題文と正解コードをもとに, 専門家により 3 段階の細目標に分類させた. その結果, 細目標が Apply の問題数は Remember および Understand の問題数を大きく上回っており, 細目標ごとの問題数に偏りが生じていた. このままでは, MMoE の細目標分類が Apply に偏って学習されるおそれがある. そこで, 各細目標の問題数の偏りを緩和するために, Apply の問題をアンダーサンプリングした. 以上の処理を経て, 最終的なデータセットの総問題数は 1079 問となった. 最終的なデータセットにおける各タグごとの問題数を表 1 に, 各細目標ごとの問題数を表 2 に示す.

表 1: 各タグごとの問題数

タグ（出題領域名）	問題数
implementation	670
data structures	73
greedy	309
math	383
dp	78

表 2: 各細目標ごとの問題数

細目標	問題数
Remember	286
Understand	469
Apply	324

表 3: MMoE とシングルタスクでの精度比較

	RMSE(Difficulty)	ACC(細目標)
シングルタスク (難易度)	0.3920(0.0348)	-
シングルタスク (細目標)	-	0.6265(**0.0454**)
MMoE	**0.3869**(**0.0247**)	**0.6283**(0.0505)

4.2 MMoE の予測精度評価

4.2.1 実験設定

MMoE の予測精度を評価し, シングルタスクモデルに対する優位性を示す. 本実験では 10 分割交差検証を行い, 訓練データ：検証データ：テストデータ=8：1：1 で分割した. 検証データは深層学習手法における early-stopping[26] で用い, 最大 100epoch として検証損失が 10 回連続で改善されなかった場合に学習を終了した. ハイパーパラメータの探索（グリッドサーチ）は, MMoE のエキスパート層およびタワー層の数を 1〜5, エキスパート数を 1〜5, 損失重み係数 λ を 0.1 から 0.9 までの 0.1 刻みの範囲で行った. 探索の結果, エキスパート層およびタワー層の数は 1, エキスパート数は 1, 損失重み $\lambda = 0.7$ を最適なハイパーパラメータとして決定した. 比較対象として, $\lambda = 1.0$ として難易度のみを学習させたシングルタスクモデル（シングルタスク・難易度）と, $\lambda = 0.0$ として細目標のみを学習させたシングルタスクモデル（シングルタスク・細目標）を用い, いずれもエキスパート数を 1 とした.

4.2.2 実験結果

表 3 に実験結果を示す. 評価指標として, 難易度は RMSE を, 細目標は Accuracy を用いた. 表

3 より, MMoE を用いることで, それぞれのタスクを個別に学習させた場合と比較して, 難易度予測の RMSE および細目標分類の Accuracy の双方が向上した. これは, MMoE のゲート機構がタスク間の負の転移を抑制しつつ, 有益な情報を共有できた可能性を示唆している. 以降の強化学習実験では, 学習済み MMoE を生成された問題の難易度および細目標の予測モデルとして用いる.

4.3 問題生成手法の比較評価

4.3.1 実験設定

本実験では, 提案手法による問題生成能力を評価するため, 以下の 11 種類の手法で比較実験を行う.

1. Few-shot プロンプト法
2. SFT（Supervised Fine-Tuning）法 [14, 15]
3. DPO (Direct Preference Optimization) 法 [16]
4. Sarkar ら [17] の強化学習手法
5. GPT-4o-mini（難易度指定）
6. GPT-4o-mini（難易度・細目標指定プロンプト 1 による難易度・細目標指定）
7. GPT-4o-mini（難易度・細目標指定プロンプト 2 による難易度・細目標指定）
8. 提案手法 1（難易度・細目標指定プロンプト 1 を用いた難易度のみのシングルタスク強化学習）
9. 提案手法 2（難易度・細目標指定プロンプト 2 を用いた難易度のみのシングルタスク強化学習）
10. 提案手法 3（難易度・細目標指定プロンプト 1 を用いた難易度・細目標のマルチタスク強化学習）
11. 提案手法 4（難易度・細目標指定プロンプト 2 を用いた難易度・細目標のマルチタスク強化学習）

Few-shot プロンプト法では, 指定難易度に近い学習データ中の問題をプロンプトに入力し, LLM に問題生成させる. SFT 法および DPO 法では, 指定難易度を含む指示プロンプトを入力して問題生成を行う. なお, DPO 法は SFT 学習済みモデルに追加学習を行う. Sarkar らの強化学習手法は, 報酬を二乗誤差 $r(\boldsymbol{x}) = -(\hat{d}(\boldsymbol{x}) - d)^2$ として難易度のみを最適化する. GPT-4o-mini（難易度指定）では, 高性能 LLM である GPT-4o-mini の API に対して指定難易度のみを含む指示プロンプトを入力して問題生成を行う. GPT-4o-mini（難易度・細目標指定プロンプト 1 による

難易度・細目標指定）および GPT-4o-mini（難易度・細目標指定プロンプト 2 による難易度・細目標指定）では, 高性能 LLM である GPT-4o-mini の API に対してそれぞれ指定難易度と指定細目標の双方を含む難易度・細目標指定プロンプト 1, または難易度・細目標指定プロンプト 2 を入力して問題生成を行う. 提案手法 1 および提案手法 2 は, 提案手法の枠組みで報酬関数の係数を $\mu = 1.0$ とし, 難易度・細目標指定プロンプト 1, または難易度・細目標指定プロンプト 2 を LLM に入力して難易度のみを最適化する. 提案手法 3 および提案手法 4 は, 難易度・細目標指定プロンプト 1, または難易度・細目標指定プロンプト 2 を LLM に入力して難易度および細目標の双方を最適化する. 提案手法 3 および提案手法 4 の報酬関数の係数は, 提案手法 4 において生成された問題の指定細目標の予測確率の平均値が最大となった $\mu = 0.8$ を採用した.

比較実験では出題領域を「implementation」に固定した. また, Sarkar ら [17] の強化学習手法, 難易度のみのシングルタスク強化学習, および提案手法では, 同一条件で 3 回の強化学習を行い, その平均値を用いた. 難易度のみのシングルタスク強化学習, および提案手法の報酬関数のパラメータ σ については, $\sigma \in \{0.2, 0.3, 0.4, 0.5\}$ の範囲で探索を行った. なお, これら 3 手法の強化学習中の評価は LLM の最尤出力に基づく.

難易度のみのシングルタスク強化学習, および提案手法においては, 生成された問題の予測難易度と指定難易度の差が 0.10 以下, かつ指定細目標の予測確率が他の細目標の予測確率を上回った時点で学習を終了した. 一方, Sarkar ら [17] の強化学習手法は, 生成された問題の予測難易度と指定難易度の差が 0.10 以下になった時点で学習を終了した. ただし 3 手法とも, 1000 ステップ到達時, または 50 ステップ連続で出力形式を満たさなかった場合は学習を打ち切った. 打ち切り時は, 難易度のみのシングルタスク強化学習, および提案手法では, 生成された問題の予測難易度と指定難易度の差が最小（複数ある場合は指定細目標の予測確率が最大）の問題を最適な生成問題とした. Sarkar ら [17] の強化学習手法では, 生成された問題の予測難易度と指定難易度の差が最小の問題を最適な生成問題とした.

提案手法においては, 指定条件のトレーニング・データ数が十分でないために強化学習が収束しない指定条件は除外した. その結果, 指定難易度は以下の通りとなった.

- **指定細目標 Remember：** -0.3〜0.0 （0.1 刻み）
- **指定細目標 Understand：** -0.1〜1.0 （0.1 刻み）
- **指定細目標 Apply:** 0.5〜1.0（0.1 刻み）

以降の問題生成手法の比較評価では, 上記の指定条件のみを対象に集計した.

4.3.2 実験結果

前節で定めた範囲における各手法で生成された問題の予測難易度と指定難易度の差の平均値と標準偏差を表 4 に, 生成された問題の指定細目標の予測確率の平均値と標準偏差を表 5 に示す.

表 4: 生成された問題の予測難易度と指定難易度の差の平均値と標準偏差

	Few-shot プロンプト法	SFT 法	DPO 法	Sarkar ら [17] の 強化学習手法	GPT-4o-mini (難易度指定)	GPT-4o-mini (難易度・細目標指定 プロンプト 1 による 難易度・細目標指定)	GPT-4o-mini (難易度・細目標指定 プロンプト 2 による 難易度・細目標指定)	提案手法 1	提案手法 2	提案手法 3	提案手法 4
平均値	0.398	0.462	0.464	0.145	0.502	0.352	0.489	0.045	**0.037**	0.039	**0.037**
標準偏差	0.267	0.282	0.262	0.130	0.285	0.263	0.351	0.022	0.026	**0.021**	0.022

表 5: 生成された問題の指定細目標の予測確率の平均値と標準偏差

	GPT-4o-mini (難易度・細目標指定 プロンプト 1 による 難易度・細目標指定)	GPT-4o-mini (難易度・細目標指定 プロンプト 2 による 難易度・細目標指定)	提案手法 1	提案手法 2	提案手法 3	提案手法 4
平均値	0.4827	0.4877	0.6795	0.6480	0.6616	**0.7150**
標準偏差	0.2267	0.2864	0.1420	0.1700	**0.0856**	0.0984

表 4 より, Few-shot プロンプト法, SFT 法, DPO 法, GPT-4o-mini（難易度指定）, GPT-4o-mini（難易度・細目標指定プロンプト 1 による難易度・細目標指定）, GPT-4o-mini（難易度・細目標指定プロンプト 2 による難易度・細目標指定）は, Sarkar ら [17] の強化学習手法および提案手法 1〜4 と比較して, 生成された問題の予測難易度と指定難易度の差の平均値が大きかった. これは, 前者が予測難易度と指定難易度の差に基づいて LLM を直接更新しないのに対し, 後者は予測難易度と指定難易度の差に基づく報酬で LLM を直接最適化するためである.

また, 表 4 より, Sarkar らの強化学習手法に比べて提案手法 1〜4 は, 予測難易度と指定難易度の差が小さかった. これは, Sarkar ら [17] の強化学習手法では二乗誤差に比例する報酬を用いるのに対し, 提案手法 1〜4 ではガウスカーネル型の報酬（式 3）を用いるため, 指定難易度近傍での予測難易度との差をより小さくする学習が促された可能性がある. 加えて, 提案手法 1〜4 は標準偏差も小さく, 条件間での難易度調整精度が安定していた.

さらに, 提案手法 1〜4 を比較すると, 予測難易度と指定難易度の差は同等であり, 細目標を同時に最適化しても難易度調整精度が大きく損なわれないことが分かる. 一方, 表 5 より, 提案手法 4 で生成された問題の指定細目標の予測確率が提案手法 1〜3, GPT-4o-mini（難易度・細目標指定プロンプト 1 による難易度・細目標指定）, GPT-4o-mini（難易度・細目標指定プロンプト 2 による難易度・細目標指定）で生成された問題の指定細目標の予測確率より高かった.

以上より, 指定細目標の詳細な定義をプロンプトに入力し, 難易度と細目標の双方を同時に最適化することで, 難易度調整精度を保ったまま細目標の達成度を向上できることが示された. この理由として, まず難易度・細目標指定プロンプト 2 では, 各細目標に対して要求される思考過程や許容される解法の性質を, 問題文の解釈, 解法方針の選択, 制御構造の複雑さといった観点から具体的に明示している点が挙げられる. これにより, LLM は問題生成時に, 指定細目標に対応する解法構造や問題設定を出力しやすくなり, 細目標に関する制約の下で問題を生成するよう誘導される. さらに, 提案手法 4 では, このようなプロンプト上の制約を与えたまま, 細目標報酬 $r_{\mathrm{bloom}}(\boldsymbol{x})$ により指定細目標に整合した問題ほど高い報酬を与える. そのため, プロンプトで規定した細目標の特徴に沿う問題の生成が強化学習を通じて逐次強化される. 一方で, 提案手法 4 では難易度報酬 $r_{\mathrm{diff}}(\boldsymbol{x})$ も同時に最適化しているため, 細目標の達成度が高い場合であっても, 指定難易度から大きく外れた問題は高い報酬を得られない. その結果, 難易度調整精度を維持したまま, 指定細目標により整合した問題を生成するように LLM が更新されたと考えられる.

4.4 提案手法で生成された問題の例

ここでは, 提案手法 4 が出力する問題文・正解コードの生成例を指定細目標ごとに掲載する.

■指定細目標が Remember **の問題例**　以下に, 指定細目標 Remember, 指定難易度-0.30 で生成された問題を示す.

```
**Question Text:**
Implement a function to check if a given number is even. The function
should take an integer as input and return a boolean value indicating
whether the number is even.

**Correct Code:**
‘‘‘python
def is_even(n):
    return n % 2 == 0
‘‘‘
```

上記の問題は剰余演算による単一の条件判定（`n % 2 == 0`）のみで解ける基礎的な問題であり, 複数手順の設計や情報統合を要しない. したがって, 基本的な知識の想起により解答可能な Remember に相当すると考えられる.

■指定細目標が Understand **の問題例**　以下に, 指定細目標 Understand, 指定難易度 0.40 で生成された問題を示す.

```
**Question Text:**
Given an array of integers, implement a function that determines whether
there exists a pair of elements in the array that adds up to a given
target sum. The function should return True if such a pair exists, and
False otherwise.

**Correct Code:**
‘‘‘python
def has_pair_sum(nums, target):
    seen = set()
    for num in nums:
        complement = target - num
```

```
        if complement in seen:
            return True
        seen.add(num)
    return False
```

上記の問題は, 配列中の各要素について, 和が target となる相手の値が既に現れているかを set を用いて順に確認することで解くことができる. このように, 問題文の要求を素直な処理手順へ翻訳し, 典型的なデータ構造を用いて実装すれば解答可能であり, 制約に基づく複雑なアルゴリズム選択は要求されないため, Understand に相当すると考えられる.

■指定細目標が Apply の問題例　以下に, 指定細目標 Apply, 指定難易度 1.00 で生成された問題を示す.

```
**Question Text:**
Design an algorithm to find the minimum and maximum feasible integer T for a
given set of integers {a1, a2,..., an} such that for every pair of distinct
elements ai and aj, the absolute difference |ai - aj| is greater than or
equal to T. The constraint is that T should be minimized while ensuring the
feasibility of the given set.

**Correct Code:**
```python
def min_max_feasible_T(nums):
    nums.sort()
    min_T = float('inf')
    max_T = float('-inf')

    for i in range(1, len(nums)):
        if nums[i] - nums[i-1] < min_T:
            min_T = nums[i] - nums[i-1]
        if nums[i] - nums[i-1] > max_T:
            max_T = nums[i] - nums[i-1]
```
```

```
    return min_T, max_T

nums = [1, 2, 3, 4, 5]
min_T, max_T = min_max_feasible_T(nums)
print("Minimum feasible T:", min_T)
print("Maximum feasible T:", max_T)
```

上記の問題では, 整数集合に対する条件 $|a_i - a_j| \geq T$ を, 単純に全ての要素対について逐一判定するのではなく, まず配列を整列し, 差分に着目して解こうとしている. このように, 問題文中の条件をそのまま機械的に翻訳するのではなく, 整列後の構造を利用して解法を設計している点で, Understand の例よりも高い認知的処理を要すると考えられる. したがって, 提案手法 4 は Apply に対応する解法設計を一定程度反映した問題を生成する傾向を示していると考えられる. 一方で, 生成された問題文と正解コードの対応にはなお改善の余地があり, Apply の問題をより安定して生成することは今後の課題である.

4.5 生成された問題の正しさの評価

強化学習により生成された問題が出題可能であることを確認するため, 生成された問題（問題文・正解コード）に対して次の評価を行った.

(i) 問題文の破綻の有無：言語モデルに基づく Perplexity（PPL）[8, 27, 28] を算出し, PPL が極端に高い問題文を除外した.

(ii) 正解コードの構文的正しさ：`ast.parse` により構文解析できないコードを除外した.

(iii) 正解コードの実行可能性：Python インタプリタで実行し, 例外が発生するコードを除外した.

(iv) 問題文と正解コードの整合性：LLM を判定器として用い, 正解コードが問題文の要求を満たさないと判定されたペアを除外した.

以降の評価では, 上記の (i)〜(iv) の評価で除外されなかった生成物を評価対象とした.

4.6 生成された問題の計量的評価

本節では, 提案手法 4 が生成した正解コードに着目し, (i) 規模（行数）と (ii) 構造的複雑さ（認知的複雑度）を評価する. また, 指定難易度および指定細目標との関係を分析する.

4.6.1 実験設定

評価対象は提案手法 4 で生成された正解コードとする．提案手法 4 では各条件について強化学習を 3 回実行しており, 各回で得られた正解コードから指標を算出する．以降では, 3 回の実行結果を条件ごとに平均して集計する．本節で用いる指標は, (i) 正解コード行数（空行および#から始まるコメント行を除く）と, (ii) 認知的複雑度（Cognitive Complexity）[29] の 2 種類である．ここで, 認知的複雑度は生成された正解コードの理解負荷に関係する構造的複雑さを評価する指標である．本指標により, 細目標（Remember/Understand/Apply）によって生成される正解コードの制御構造を中心とする構造的複雑さの違いを分析する．サイクロマティック複雑度が分岐に基づく独立経路数（テスト観点）を主に捉える指標である [30] ことに対し, 認知的複雑度は, 制御フローを読み進めて理解する際の困難さに着目して設計された指標である [29]．SonarSource の説明では, 認知的複雑度は主に以下の方針で加点される [29].

1. 条件分岐・ループ・例外処理・ジャンプ・複合条件（複数演算子を含む条件）など, 直線的な読み進めを妨げる制御構造が現れるたびに加点される．
2. ネストが深くなるほど追加で加点される．
3. 通常のメソッド呼び出しは加点されないが, 再帰呼び出しは加点対象となる．

このため, 認知的複雑度はコード行数と比べて, 分岐やネストの深さといった理解負荷に直結する構造の変化を反映しやすい.

4.6.2 正解コードの行数

図 3 に, 生成された正解コードの行数の平均値を示す．図 3 より, 指定難易度が高くなるほどコード行数が増加する傾向が確認できる．また, ブルーム・タキソノミーの細目標が高度になるほど, コード行数が増加する傾向が見られた.

4.6.3 正解コードの認知的複雑度

図 4 に認知的複雑度の平均値を示す．図 4 より, 指定細目標ごとに認知的複雑度の水準に差が見られる．具体的には, Remember では認知的複雑度が全条件で 0〜2 程度にとどまっており, 複雑な分岐や反復をほとんど含まない単純なコードが生成されていることが分かる．これは, 制御構造を多用しない正解コードを要求するプロンプト上の制約と整合している．一方, Understand では認知的複雑度がおおむね 3〜6 程度となっており, Remember より高い値を示した．これは, 問題文の要求を複数の処理手順へ落とし込む正解コードや, 典型的なデータ構造や走査処理を組み合わ

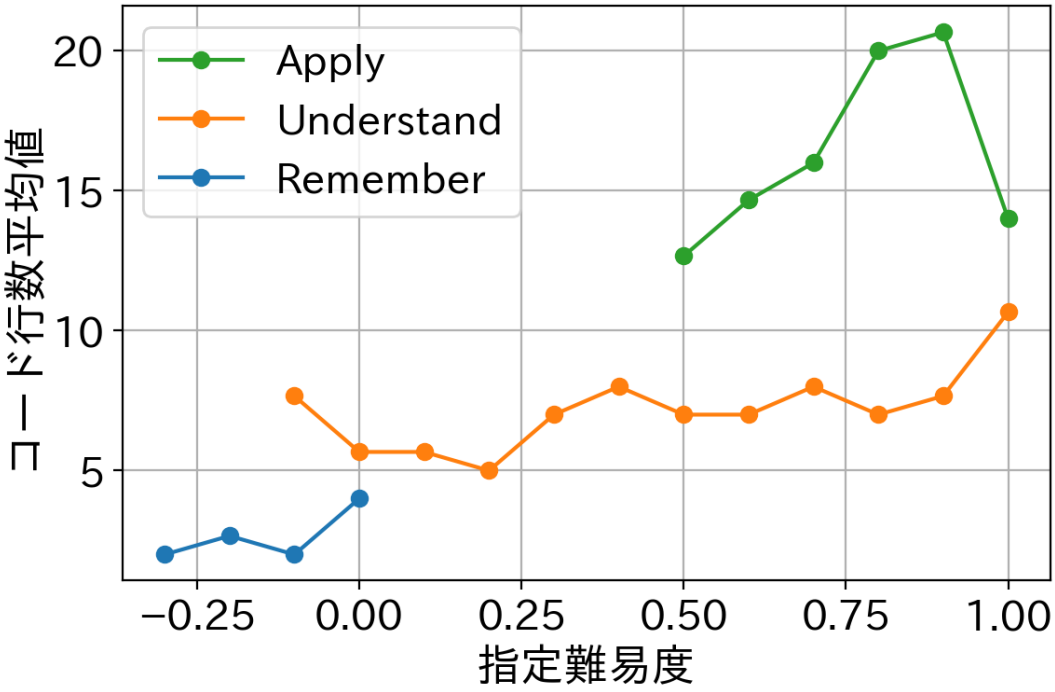

図 3: コード行数の平均値

せるコードが生成されやすかったことを示唆している．さらに, Apply では認知的複雑度がおおむね 4〜9 程度となり, 多くの条件で Understand を上回った．これは, Apply に対してアルゴリズム選択や解法設計を要する問題を求めるプロンプトを与えた結果, より多くの制御構造や処理手順を含むコードが生成されやすくなったためと考えられる．以上より, 認知的複雑度の結果は, 細目標ごとにプロンプトで与えた解法上の制約や要求の違いが, 生成された正解コードの構造的複雑さにも概ね反映されていることを示している．

4.7 生成された問題のエキスパート評価および GPT-4o-mini の API 評価

本節では, 提案手法 4 で生成された問題文と正解コードが指定条件（難易度・細目標）にどの程度合致しているかを確認するため, 人間のエキスパート評価および GPT-4o-mini の API による評価を実施する．

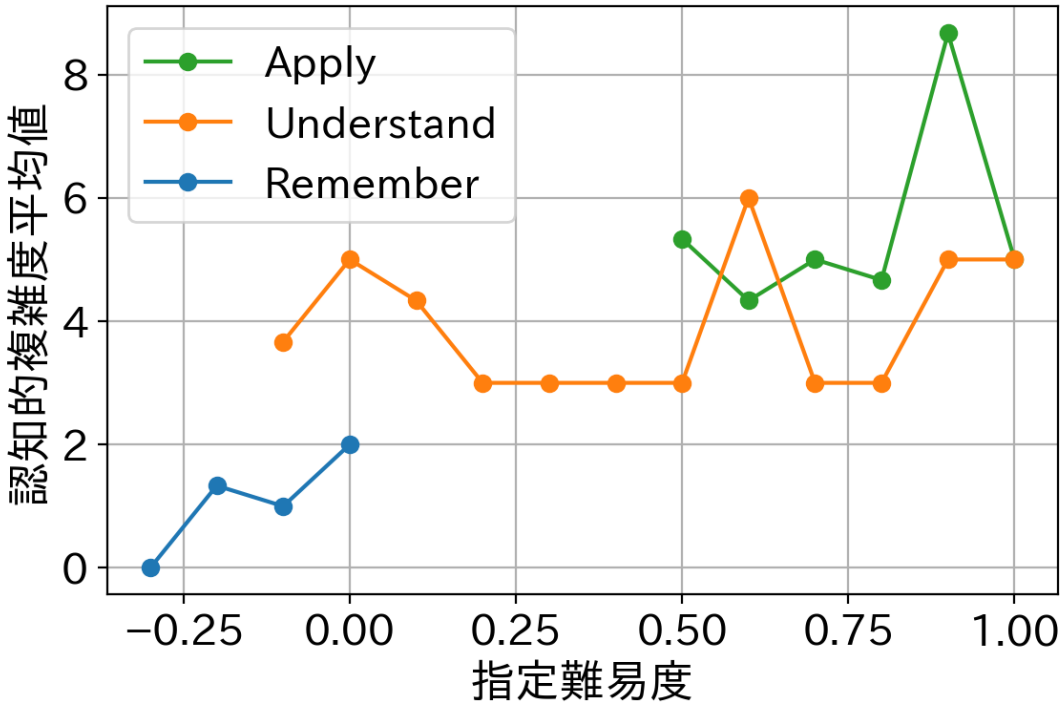

図 4: 認知的複雑度の平均値

4.7.1 実験設定

評価対象は提案手法 4 で生成された問題文と正解コードとする．提案手法 4 では各条件について強化学習を 3 回実行しており, 各回で得られた生成問題を評価する．以降では, 難易度については各生成問題に対して得られた評価値を用いて条件ごとに平均を集計し, 細目標については各生成問題に対して決定された評価ラベルを用いて指定細目標ごとの割合を集計する．

評価は, (i) 難易度（5 段階: 1〜5）と, (ii) 細目標（Remember/Understand/Apply）の評価の 2 種類とした．エキスパート評価では 5 人のエキスパートが独立に評価した値を用いる．難易度については, 5 人の評価の平均を各生成問題の評価値とする．一方, 細目標については, 5 人の評価の多数決により各生成問題の評価ラベルを決定する．

表 6: エキスパート評価の多数決により決定した細目標ラベルの割合

指定細目標＼エキスパート評価 細目標	Remember	Understand	Apply
Remember	1.00	0.00	0.00
Understand	0.08	0.92	0.00
Apply	0.00	0.28	0.72

4.7.2 エキスパートによる評価

本節では, 提案手法 4 で生成された問題を, プログラミング教育を専門とする大学教員（エキスパート）5 名に評価してもらった. エキスパートによる評価では, 難易度を 1〜5 の 5 段階で, 細目標を Remember, Understand, Apply の 3 段階で評価してもらった. まず, 難易度については, 非常に簡単な問題を 1, 簡単な問題を 2, 標準的な問題を 3, 難しい問題を 4, 非常に難しい問題を 5 として評価してもらった.

次に, 細目標については,

- Remember： 文法・演算・基本的な API を暗記していれば解答可能であり, 正解コード中の制御構造（for/while/if）はいずれか 1 つまでの問題
- Understand： 仕様を正しく読み取り, 手順・条件を整理して実装すれば解答可能な問題. 要求される手法は典型的であり, 制約によるアルゴリズムの選択を要求しない問題
- Apply： 制約により単純な解法が成立せず, 効率を考慮した発想やアルゴリズム選択が必要となる問題

で評価してもらった.

図 5 にエキスパートにより評価された難易度の平均値を示す. 図 5 より, 全体として, 指定難易度が高くなるにつれてエキスパート評価の難易度も上昇する傾向が見られた. また, 指定細目標ごとにエキスパート評価の難易度にも差が見られ, Remember が最も低く, Understand がそれに続き, Apply が最も高い値を示した. すなわち, 生成された問題は指定難易度だけでなく, 指定細目標の違いも反映して, エキスパートから異なる難しさとして評価されていることが示唆される.

表 6 に, 5 人のエキスパート評価の多数決により決定した細目標ラベルの割合を, 各指定細目標ごとに示す. 表 6 より, 指定細目標が Remember の場合は多数決の結果, すべての問題が Remember と評価されており, Remember の生成は安定して行えていることが分かる. また, 指定細目標が Understand の場合は多数決の結果, 92% の問題が Understand, 8% の問題が

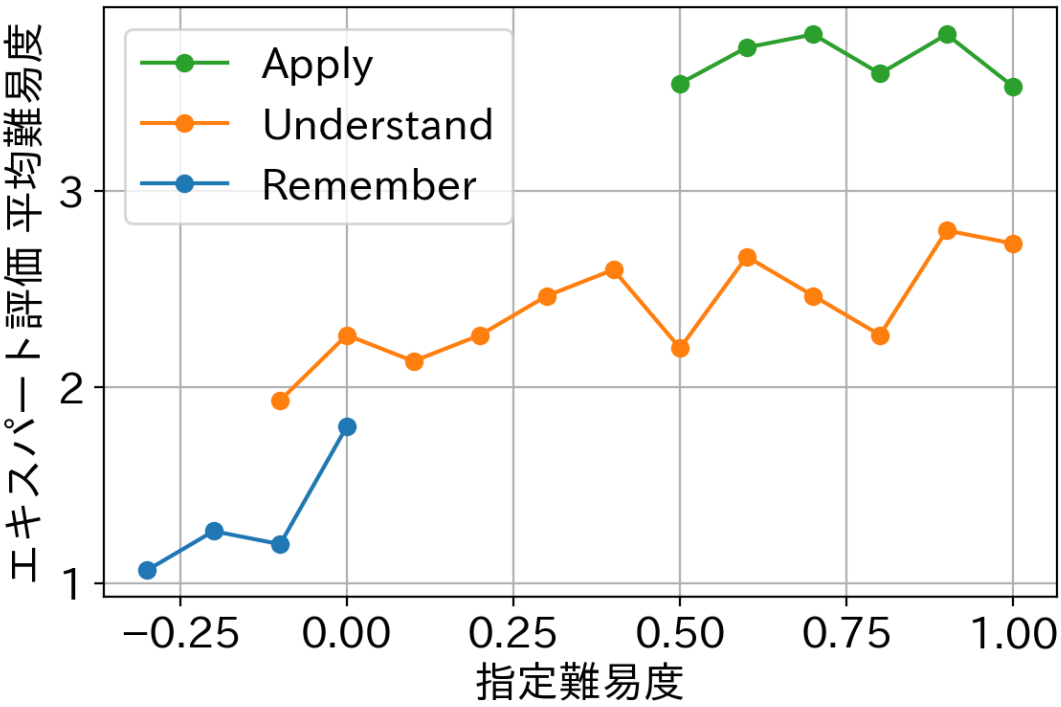

図 5: エキスパートにより評価された難易度の平均値

Remember と評価されており, 大半の問題で指定細目標を反映した生成ができている. さらに, 指定細目標が Apply の場合は多数決の結果, 72% の問題が Apply, 28% の問題が Understand と評価されており, Apply に対応する問題を一定程度生成できている一方で, 一部の問題は Understand と評価されている. 以上より, 提案手法 4 は細目標を概ね反映した問題生成ができていると考えられるが, 特に Apply に対応する問題をより安定して生成することにはなお改善の余地がある.

4.7.3 GPT-4o-mini の API による評価

本節では, 高性能 LLM である GPT-4o-mini の API により, 提案手法 4 で生成された問題の難易度・細目標を評価し, 分析する. GPT-4o-mini の API による評価については, 提案手法 4 で生成された各問題（問題文と正解コードのペア）に対し, 以下のプロンプトで GPT-4o-mini の API を用いて難易度を評価した. 問題文と正解コードを入力として与え, 5 段階（1〜5）の難易度を JSON 形式で出力させた. 難易度評価では, コード長のみに依存せず, 問題文の理解に必要な推論量, 要求される手法の難しさ, 制約が解法選択に与える影響, および場合分けや境界条件処理の多さを総合的に判断するよう指示した. 難易度は次の基準で評価させた：

- Level 1： 基本規則・構文・単純な式を直接適用するだけで解くことができ, 分岐や場合分けをほとんど要しない問題
- Level 2： 基本構文 1 つ（単純な if, ループ, 算術・文字列処理等）を用いた素直な処理で解け, 複雑な解法選択を要しない問題
- Level 3： 問題文の要求を複数の処理手順へ分解して実装する必要があり, 軽い場合分けや典型的なデータ構造・パターンを用いる問題
- Level 4： 典型的なテクニック（例：ソートと走査, two-pointers, greedy, prefix sums 等）や, 境界条件処理を含む注意深い設計を要する問題
- Level 5： 制約により単純な解法が成立せず, DP, グラフ探索, feasibility check を伴う二分探索等の高度な手法や重い推論を要する問題

また, 問題文と正解コードの両方を根拠として判断するよう指示し, 併せて確信度（0〜1）と短い理由（25 語以内）を出力させた. 具体的には, 以下のプロンプトで難易度を評価させた.

```
You are a strict evaluator for Python programming problems.

Return JSON only:
{
  "difficulty_level": 1|2|3|4|5,
  "confidence": 0.0-1.0,
  "reason": "short (<= 25 words)"
}

Judge difficulty based on BOTH the problem statement and the reference solution code.
Do NOT decide difficulty from code length alone. Short code can still be difficult.
Pay attention to the required reasoning, the need for algorithm selection, the role
of constraints, and the amount of case handling.

Choose ONE difficulty level using these criteria:

Level 1 (Very Easy):
- Solvable by direct application of one basic rule, syntax, or formula.
- Almost no case analysis, no meaningful algorithm choice.
```

- Typically direct print / arithmetic / one obvious check.

Level 2 (Easy):
- Solvable by one straightforward basic construct or one simple procedure.
- Minimal decomposition; little risk of mistakes.
- No nontrivial algorithm selection is required.

Level 3 (Moderate):
- Requires interpreting the statement and translating it into multiple straightforward steps.
- May use a standard data structure or a typical pattern, but the method is still direct.
- Some case handling is needed, but constraints do not force a difficult algorithmic choice.

Level 4 (Hard):
- Requires a clear technique beyond direct translation.
- Constraints or problem structure require careful method selection, nontrivial case handling, or structured reasoning.
- Typical examples include sorting + scan, two-pointers, greedy pattern, prefix sums, or structured simulation with pitfalls.

Level 5 (Very Hard):
- Requires advanced algorithmic design or heavy reasoning.
- Constraints strongly rule out naive approaches.
- Typical examples include DP, graph algorithms, binary search on answer with feasibility check, or other advanced methods.

Rules:
- Use the statement and the code together.
- Consider constraints if they affect method choice.
- If the code uses an advanced method but the statement does not really require it,

```
do not overrate difficulty.
- If uncertain between two adjacent levels, choose the higher one with lower
confidence.
```

また, 細目標についても同様に, 各問題に対して細目標（Remember/Understand/Apply）を評価した. 細目標については, 問題文と正解コードの両方に基づいて 3 分類のラベルを JSON 形式で出力させた. エキスパート評価と同様に, 細目標は次の基準で評価させた：

- Remember： 文法・演算・基本的な API を暗記していれば解答可能であり, 正解コード中の制御構造（for/while/if）はいずれか 1 つまでの問題
- Understand： 仕様を正しく読み取り, 手順・条件を整理して実装すれば解答可能な問題. 要求される手法は典型的であり, 制約によるアルゴリズムの選択を要求しない問題
- Apply： 制約により単純な解法が成立せず, 効率を考慮した発想やアルゴリズム選択が必要となる問題

具体的には, 以下のプロンプトで細目標を評価させた.

```
You are a strict evaluator for Python programming problems.

Task:
Return ONLY the cognitive target in Bloom's taxonomy (Remember/Understand/Apply).
Do NOT output any difficulty number or words like easy/hard.

Return JSON only:
{
  "cognitive_target": "Remember"|"Understand"|"Apply",
  "confidence": 0.0-1.0,
  "reason": "short (<= 25 words)"
}

Judge based on BOTH the problem statement and the reference solution code.

Definitions:
- Remember:
```

```
  Solvable by recalling and directly applying basic syntax, operators, or simple
  APIs.
  The reference solution should use at most ONE control structure among
  for / while / if, with no nesting.
  No multi-step solution design or algorithm choice based on constraints is required.

- Understand:
  Solvable by correctly interpreting the specification and organizing the required
  steps/conditions.
  The method may use a standard or typical technique, but constraints do NOT force a
  nontrivial algorithm choice.

- Apply:
  A naive direct solution is not sufficient because constraints or efficiency matter.
  Solving requires an algorithmic idea, method selection, or nontrivial solution
  design.

Rules:
- Use the statement and the code together.
- Focus on what is required to solve the problem, not only on superficial code length.
- If a standard technique is used in a direct and typical way, prefer Understand.
- Choose Apply only when constraints or efficiency genuinely require algorithm
selection or solution design.
- If uncertain between adjacent labels, choose the lower label with lower confidence.
```

図 6 に GPT-4o-mini の API により評価された難易度の平均値を示す．図 6 より, 指定細目標ごとに GPT-4o-mini の API による難易度評価に差が見られた．具体的には, Remember を指定した条件では難易度評価が一貫して 1 付近と低く, Understand ではおおむね 2〜3 程度, Apply ではおおむね 3〜4 程度の値を示した．すなわち, GPT-4o-mini の API は, 指定細目標の違いを反映して, Remember, Understand, Apply の順に高い難易度を付与する傾向を示した．また, Understand および Apply を指定した条件では, 指定難易度の上昇に伴って難易度評価も全体として上昇する傾向が見られた一方で, 一部の条件では単調な増加関係から外れる点も見られた．以

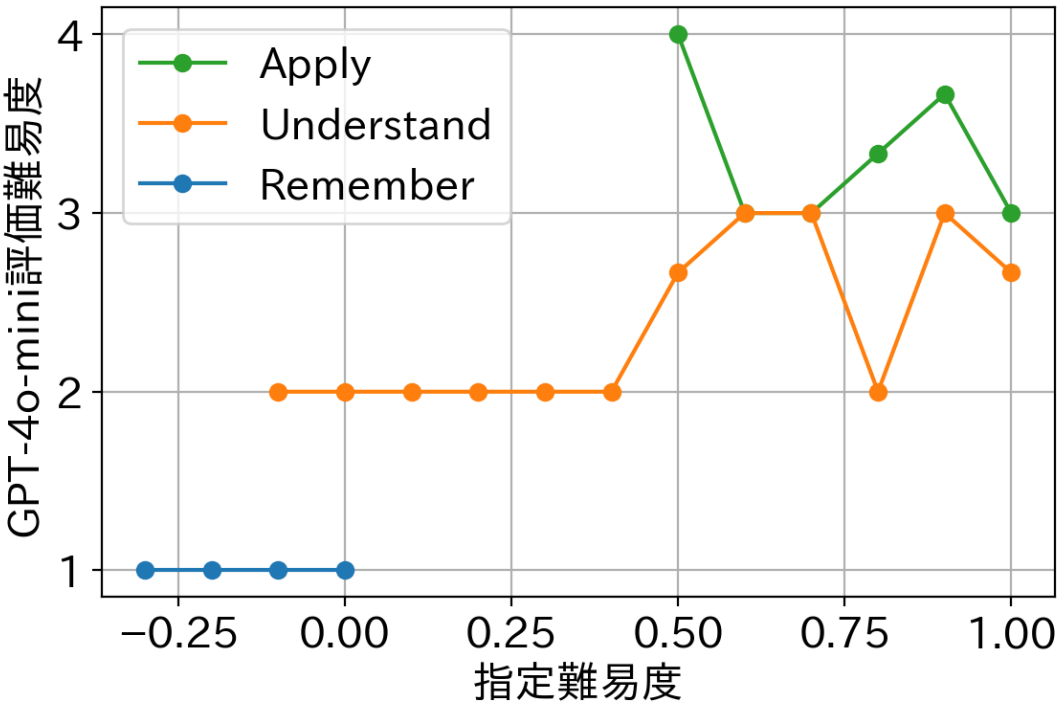

図 6: GPT-4o-mini の API により評価された難易度の平均値

表 7: GPT-4o-mini の API により評価された細目標ラベルの割合

指定細目標＼GPT-4o-mini 評価 細目標	Remember	Understand	Apply
Remember	0.67	0.33	0.00
Understand	0.00	0.28	0.72
Apply	0.00	0.00	1.00

上より, GPT-4o-mini の API による評価においても, 指定難易度および指定細目標の違いが一定程度反映されていることが示唆される.

表 7 に GPT-4o-mini の API により評価された細目標ラベルの割合を, 各指定細目標ごとに示す. 表 7 より, 指定細目標が Remember の場合は 67% が Remember, 33% が Understand と評価されており, Remember の問題の多くは正しく評価されているものの, 一部は Understand とみなされている. また, 指定細目標が Understand の場合は 28% が Understand, 72% が Apply と評価されており, GPT-4o-mini の API は Understand の問題の多くを Apply として評価する傾向を示した. さらに, 指定細目標が Apply の場合は全ての問題が Apply と評価されており,

Apply の問題は安定して Apply と認識されている．以上より, GPT-4o-mini の API は Apply に対応する問題を強く Apply と評価する一方で, Remember および Understand に対応する問題については, エキスパート評価とは異なる判定傾向を示すことが分かる．

4.7.4 エキスパート評価と GPT-4o-mini の API による評価の比較

生成結果が指定条件にどの程度従っているかを定量化するため, 指定難易度とエキスパート／GPT-4o-mini の API が評価した難易度との Pearson 相関係数を比較した．また, 細目標については, エキスパート評価および GPT-4o-mini の API 評価によって得られた細目標ラベルの割合を比較した．

表 8: 難易度評価の相関係数の比較

指定細目標	GPT-4o-mini の API による評価	エキスパート評価
Remember	—	0.855
Understand	0.686	0.748
Apply	-0.338	-0.036
All	0.800	0.794

表 8 に難易度評価の相関係数を示す．なお, 指定細目標が Remember の場合は, GPT-4o-mini の API による難易度評価がほぼ一定値を示したため, Pearson 相関係数は算出できなかった．一方, エキスパート評価は高い正の相関を示しており, Remember 条件では指定難易度との対応をより強く反映していたと考えられる．指定細目標が Understand の場合も, エキスパート評価の相関係数が GPT-4o-mini の API による評価を上回っており, 指定難易度との対応をより強く反映していた．また, 指定細目標が Apply の場合は両者とも負の相関を示したが, エキスパート評価の方が 0 に近く, GPT-4o-mini の API による評価よりも指定難易度との乖離が小さいと解釈できる．一方, 全細目標をまとめた場合（All）では, GPT-4o-mini の API による評価の相関係数がエキスパート評価をわずかに上回った．ただし, 全細目標をまとめた相関係数には細目標間の評価難易度の差の影響も含まれると考えられる．したがって, 全細目標をまとめた場合の相関のみから GPT-4o-mini の API による評価の方が優れているとは言えず, 細目標ごとの相関係数の結果も踏まえると, 難易度追従性の検証においてはエキスパート評価の方がより信頼できる基準となり得ることが示唆される．

細目標の評価については, 表 6 および表 7 を比較すると, エキスパート評価では指定細目標が Remember の場合は全て Remember, Understand の場合は 92% が Understand, Apply の

場合は 72% が Apply と判定されており, 指定細目標を概ね反映した結果となっていた. 一方, GPT-4o-mini の API による評価では, 指定細目標が Remember の場合は 67% が Remember, 指定細目標が Understand の場合は 28% が Understand, 指定細目標が Apply の場合は全て Apply と判定されていた. すなわち, GPT-4o-mini の API は Apply に対応する問題については安定して Apply と評価できており, Remember に対応する問題の多くも Remember と評価していることから, 細目標の違いを一定程度反映していることが分かる. ただし, Remember および Understand に対応する問題では, エキスパート評価と一致しない判定も一定数見られた. 特に, Understand に対応する問題では Apply と評価される割合が高く, GPT-4o-mini の API は細目標をエキスパート評価よりも高めに判定する傾向が示唆される. よって, 細目標の評価に関しては, GPT-4o-mini の API も一定の有用性があるものの, エキスパート評価の方がより信頼性の高い基準となり得ることが示唆される.

以上より, 難易度および細目標の追従性の検証において, 全体としてはエキスパート評価の方が指定条件との整合をより強く反映できることが示唆された. これは, 生成結果の追従性検証においてエキスパート評価がより信頼できる基準となり得ることを示す. 一方で, GPT-4o-mini の API による評価においても, 難易度や細目標の違いを一定程度反映した傾向は確認された. 特に, 細目標の評価では Apply に対応する問題を安定して Apply と判定しており, 補助的な評価指標として一定の有用性があると考えられる. このため, 生成結果の検証では, エキスパート評価を主としつつ, GPT-4o-mini の API による評価も補助的に併用することが有効である.

5 むすび

本研究では, 難易度だけでなく細目標を指定し, 双方を満たすプログラミング問題（問題文および正解コード）を生成する強化学習手法を提案した. 提案手法では, LLM が生成した問題の難易度および細目標を同時に予測するための MMoE を導入し, その出力に基づいて難易度と細目標を同時に最適化する報酬関数を設計した. さらに, 細目標ラベルのみを与えるのではなく, 各細目標に対して要求される思考過程や許容される解法の性質を具体的に記述したプロンプトを導入することで, 細目標に関する制約を明示した問題生成を実現した.

評価実験の結果, 細目標報酬と細目標の定義を具体化したプロンプトの双方を導入した提案手法は, 難易度のみを最適化する手法, 細目標報酬を導入して難易度と細目標の双方を最適化する手法, および細目標の定義を具体化したプロンプトを導入した上で難易度のみを最適化する手法と比較して, 指定難易度と予測難易度の誤差を同程度に保ちながら, 指定細目標の予測確率を向上できることを示した.

また, 生成された正解コードの計量的分析より, 指定難易度が高くなるにつれてコード行数が増加する傾向が確認された. さらに, 指定細目標が Remember, Understand, Apply と高くなるにつれて, コード行数も増加する傾向が確認された. 認知的複雑度に着目すると, Remember では低い値にとどまり, Understand, Apply と高くなるにつれて認知的複雑度も高くなる傾向が確認された. これらの結果は, 細目標ごとにプロンプトで与えた解法上の制約や要求の違いが, 生成された正解コードの規模や構造的複雑さにも概ね反映されていることを示唆している.

さらに, エキスパート評価および GPT-4o-mini の API による評価を通じて, 生成問題が指定難易度および指定細目標にどの程度追従しているかを分析した. その結果, GPT-4o-mini の API による評価においても難易度や細目標の違いを一定程度反映した傾向は確認されたものの, 全体としてはエキスパート評価の方が指定条件との整合をより強く反映していた. 特に細目標の評価では, GPT-4o-mini の API は Understand に対応する問題を Apply と高めに評価する傾向が見られ, 細目標をエキスパート評価よりも高めに判定する傾向が示された. このことから, 生成問題の難易度および細目標の追従性の検証には, エキスパート評価を主としつつ, GPT-4o-mini の API による評価も補助的に併用することが有効であることが示唆された.

今後の課題として, MMoE を Codeforces 以外のデータセットで学習し, 難易度や細目標の分布, および問題形式が異なる条件下でも, 提案手法が難易度と細目標の双方を安定して制御できるかを検証することが挙げられる.

謝辞

本研究の遂行にあたり, 指導教員の植野真臣教授より丁寧かつ熱心なご指導を賜りました. ここに深く感謝申し上げます. また, 本研究に関して多くの貴重な助言をいただいた渕本壱真准教授に厚く御礼申し上げます. 最後に, 日頃より研究生活を支えてくださった研究室の皆様に感謝申し上げます.

参考文献

[1] Niklas Humble. Teacher observations of programming affordances for k-12 mathematics and technology. *Education and Information Technologies*, 27(4):4887–4904, 2022.

[2] Peter J Denning and Edward E Gordon. A technician shortage. *Communications of the ACM*, 58(3):28–30, 2015.

[3] 植野真臣. CBT の最前線. **情報処理**, 64(5):e1–e6, 2023.

[4] Laura Zavala and Benito Mendoza. On the use of semantic-based aig to automatically generate programming exercises. In *Proceedings of the 49th ACM technical symposium on computer science education*, pages 14–19, 2018.

[5] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43, 2022.

[6] Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language models can teach themselves to program better. In *The Eleventh International Conference on Learning Representations*, 2023.

[7] Aysa Fan, Ranran Haoran Zhang, Luc Paquette, and Rui Zhang. Exploring the potential of large language models in generating code-tracing questions for introductory programming courses. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7406–7421, Singapore, December 2023. Association for Computational Linguistics.

[8] Salima Lamsiyah, Abdelkader El Mahdaouy, Aria Nourbakhsh, and Christoph Schommer. Fine-tuning a large language model with reinforcement learning for educational question generation. In *International Conference on Artificial Intelligence in Education*, pages 424–438. Springer, 2024.

[9] Nguyen Binh Duong Ta, Hua Gia Phuc Nguyen, and Swapna Gottipati. Exgen: Ready-to-use exercise generation in introductory programming courses. In *International Conference on Computers in Education*, 2023.

[10] Kunze Li and Yu Zhang. Planning first, question second: An llm-guided method for controllable question generation. In *Findings of the Association for Computational Lin-*

guistics ACL 2024, pages 4715–4729, 2024.

[11] Kunze Li and Yu Zhang. Crossqg: Improving difficulty-controllable question generation through consistency enhancement. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 2783–2798, 2025.

[12] Sixing Wu, Jiahao Chen, Yujue Zhou, Zhijun Yang, and Wei Zhou. Asking questions with thoughts: An efficient difficulty-controllable question generation method with posterior knowledge distillation. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 3416–3426, 2025.

[13] Julien Pourcel, Cédric Colas, Gaia Molinaro, Pierre-Yves Oudeyer, and Laetitia Teodorescu. Aces: Generating a diversity of challenging programming puzzles with autotelic generative models. *Advances in Neural Information Processing Systems*, 37:67627–67662, 2024.

[14] Yuto Tomikawa, Ayaka Suzuki, and Masaki Uto. Adaptive question–answer generation with difficulty control using item response theory and pre-trained transformer models. *IEEE Transactions on Learning Technologies*, 2024.

[15] Yuto Tomikawa and Masaki Uto. Difficulty-controllable multiple-choice question generation for reading comprehension using item response theory. In Andrew M. Olney, Irene-Angelica Chounta, Zitao Liu, Olga C. Santos, and Ig Ibert Bittencourt, editors, *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky*, pages 312–320, Cham, 2024. Springer Nature Switzerland.

[16] Yuto Tomikawa and Masaki Uto. Difficulty-controllable multiple-choice question generation using large language models and direct preference optimization. *arXiv preprint arXiv:2510.19265*, 2025.

[17] Soujatya Sarkar, Manikandan Ravikiran, and Rohit Saluja. Scorecliq: A dynamic llm-based framework for item difficulty estimation. In Alexandra I. Cristea, Erin Walker, Yu Lu, Olga C. Santos, and Seiji Isotani, editors, *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium, Blue Sky, and WideAIED*, pages 169–176, Cham, 2025. Springer Nature Switzerland.

[18] Ching Han Chen and Ming Fang Shiu. Kaqg: A knowledge-graph-enhanced rag for difficulty-controlled question generation. *arXiv preprint arXiv:2505.07618*, 2025.

[19] David R Krathwohl. A revision of bloom's taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.

[20] Zehua Xia, Qi Gou, Bowen Yu, Haiyang Yu, Fei Huang, Yongbin Li, and Cam-Tu Nguyen. Improving question generation with multi-level content planning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 800–814, Singapore, December 2023. Association for Computational Linguistics.

[21] Chuyao Ding, Yu Hong, and Jianmin Yao. SGCM: Salience-guided context modeling for question generation. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 14755–14762, Torino, Italia, May 2024. ELRA and ICCL.

[22] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018.

[23] Frederic M Lord. *Applications of item response theory to practical testing problems.* Routledge, 2012.

[24] Juntae Kim, Eunjung Cho, and Dongbin Na. Problem-solving guide: Predicting the algorithm tags and difficulty for competitive programming problems, 2024.

[25] Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. `https://huggingface.co/datasets/open-r1/codeforces`, 2025.

[26] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in neural information processing systems*, 2, 1989.

[27] Sahan Bulathwela, Hamze Muse, and Emine Yilmaz. Scalable educational question generation with pre-trained language models. In *International Conference on Artificial Intelligence in Education*, pages 327–339. Springer, 2023.

[28] Zichao Wang, Jakob Valdez, Debshila Basu Mallick, and Richard G Baraniuk. Towards human-like educational question generation with large language models. In *International*

conference on artificial intelligence in education, pages 153–166. Springer, 2022.

[29] G Ann Campbell. Cognitive complexity: An overview and evaluation. In *Proceedings of the 2018 international conference on technical debt*, pages 57–58, 2018.

[30] Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. Cyclomatic complexity. *IEEE software*, 33(6):27–29, 2016.