一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

Dockerイメージの品質改善を目的とした Dockerfile最適化支援ツールの試作

森内　涼太†　　柗本　真佑†　　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{r-moriut,shinsuke,kusumoto}@ist.osaka-u.ac.jp

あらまし　軽量なコンテナ型仮想環境プラットフォームとしてDockerが注目されている．Dockerイメージの品質改善としては，Dockerfileから生成されるDockerイメージのサイズとビルド時間の削減が重要である．しかしながら，Dockerfileの最適化作業に必要なビルド時の動的情報は，コードエディタ上の静的情報であるソースコードからは把握できない．そのため，動的情報はDockerfileのエディタとは独立したターミナル上で確認する必要があり，Dockerfileの各行との対応が取りにくい．また，最適化効果の把握に必要なビルド間の差分を直接把握する方法は存在しない．本研究ではビルド時の動的情報を解析し，解析結果をコードエディタ上で閲覧できるツールを提案する．被験者実験の結果，Dockerfileの最適化作業において，正確性と効率性の向上に一定の効果があることを確認した．
キーワード　Docker，Dockerイメージ，Dockerfile，最適化，レイヤ，ビルドキャッシュ

1. はじめに

軽量なコンテナ型仮想環境プラットフォームとしてDockerが注目されている．Dockerは，一般的なハイパーバイザ型の仮想化技術と比べてリソース効率が高く，仮想環境の迅速なデプロイとスケーリングを可能とする．コンテナの作成に用いるDockerイメージはDocker Hubなどのホスティングサービスを通じて共有が可能であり，仮想環境の再現性や可搬性の高さに貢献している．このような利点から，DockerはOSSのようなソフトウェア開発プロジェクトで広く採用されている[1]．さらにソフトウェア開発のみならず，IoTやクラウドコンピューティングなど幅広い分野で活用されている[2][3]．

Dockerイメージの開発においては，一般的なソフトウェアの開発と同様，品質改善が欠かせない．具体的には，Dockerイメージに対するサイズとビルド時間の削減が挙げられる．前者はリソース効率や可搬性の向上が，後者はDockerイメージの開発効率の向上が目的である．Dockerイメージの品質改善には，Dockerイメージのソースコードである Dockerfileの修正が必須である．本稿では，このような品質改善を目的としたDockerfileの修正を最適化と表現する．

Dockerfileの最適化は，一般的なソースコードに対するバグ修正やリファクタリングと同様，容易ではない．本研究では，Dockerfileの最適化作業を「問題点の発見」「問題点の分析」「問題点の修正」「修正効果の把握」の4ステップに分解して考える．「問題点の発見」では，サイズが大きい，もしくはビルドキャッシュが利用されていないなどの問題を抱えたレイヤを発見するために，ビルド時の動的情報を把握しなければならない．一方で，Dockerfileの開発に用いるコードエディタで把握できる情報はソースコード上の静的情報に限られている．コードエディタとは独立したターミナル上で動的情報を調べる場合，動的情報とDockerfileの各行の対応は取りづらい．また「修正効果の把握」では，動的情報がDockerfileの修正前後でどのように変化したか把握しなければならない．

本研究の目的は，Dockerfile最適化作業の支援である．特に最適化作業の4ステップのうち，「問題点の発見」と「修正効果の把握」に着目する．そのためにビルド時の動的情報を解析し，解析結果をコードエディタ上で閲覧できるツールを提案する．本研究では，提案ツールの有効性を評価するために被験者実験を実施した．実験の結果，Dockerfileの最適化作業において，正確性と効率性の向上に一定の効果があることを確認した．

2. 準　　備

2.1 Dockerイメージの開発

Dockerイメージの構築手順は，Dockerfileと呼ばれるソースコード内に記述される．Dockerfileの記述例を図1に示す．図のDockerfileには，Pythonプログラムの実行環境を構築する手順が記述されている．Ubuntuの Dockerイメージをベースにパッケージや依存ライブラリのインストールを行い，Pythonプログラムをコンテナ起動時に実行できる環境を整えている．

Dockerのビルドコマンドを実行すると，Dockerfileに記述されたコマンドが上から順に実行され，各コマンドごとにレイヤが生成される．これら複数のレイヤによって1つのDockerイメージが構成される．なお，各レイヤは一度生成されると，変更されるまでビルドキャッシュされる．ビルド時に変更のないレイヤは再構築せずにビルドキャッシュが利用されるため，ビルド時間の短縮に繋がる．

2.2 Dockerfileの最適化

Dockerイメージの品質を左右するイメージのサイズとビルド時間を削減するためには，Dockerfileの最適化が必須であ

```
1  # ベースイメージを指定
2  FROM ubuntu:20.04
3  # パッケージをインストール
4  RUN apt-get update && apt-get install -y \
5      python3 python3-pip \
6      openjdk-17-jdk \
7      && rm -rf /var/lib/apt/lists/*
8  # 依存ライブラリをインストール
9  COPY . .
10 RUN pip3 install --no-cache-dir -r requirements.txt
11 # コンテナ起動時にPythonプログラムを実行
12 CMD ["python", "sample.py"]
```

図 1　Dockerfile の記述例

る．Dockerfile の最適化は Docker プロジェクトで一般的に実施されており，その注目度は高い．Docker Hub に公開されている Docker イメージのサイズは，最適化の実施により，バージョンを重ねるにつれて減少傾向にあると報告されている [4]．GitHub で管理されている Docker プロジェクトのリポジトリでは，Dockerfile に関連するコミットに，単なるリファクタリングのみならずイメージサイズやビルド時間の削減を目的とした最適化が多く含まれていると確認されている [5]．

本研究では，Dockerfile の最適化作業を「STEP1：問題点の発見」「STEP2：問題点の分析」「STEP3：問題点の修正」「STEP4：修正効果の把握」の 4 ステップに分解して考える．ここでは前述の 図 1 を例に説明する．まず各レイヤのサイズを調べた結果，4〜7 行目の RUN コマンドに対応するレイヤのサイズが，他のレイヤと比較して大きいと判明したとする．次に問題となっている RUN コマンドを分析すると，Python プログラムの実行環境の構築とは無関係な openjdk-17-jdk がインストールされていると分かる．つまり，不要なパッケージがレイヤのサイズに影響を与えている可能性が高い．この分析結果をもとに 6 行目を削除し，改めて Docker イメージのビルドを行う．最後に，RUN コマンドのレイヤのサイズを修正前後で比較し，修正効果がどの程度得られたか把握する．

2.3　Dockerfile 最適化の課題

2.2 節で述べた Dockerfile の最適化作業について，「STEP1：問題点の発見」と「STEP4：修正効果の把握」には課題が存在する．

「STEP1：問題点の発見」を実施する際には，各レイヤのサイズやビルド時間の変動，ビルドキャッシュがどのレイヤまで利用されているかといった動的情報を把握しなければならない．一方で，Dockerfile の開発に用いるコードエディタで把握できる情報はソースコード上の静的情報に限られるため，問題点の発見は困難である．また，動的情報を調べる Docker の標準コマンドや dive[1] といったツールは存在するが，コードエディタとは独立したターミナル上で調べなければならない．そのため，動的情報と Dockerfile の各行の対応が取りづらく，Dockerfile の修正箇所の特定を妨げる要因となる．

「STEP4：修正効果の把握」を実施する際には，STEP1 で調べた動的情報が Dockerfile の修正前後でどのように変化したか把握しなければならない．しかしながら，このようなビルド間の差分を直接把握する方法は我々の知る限り存在しない．そのため，開発者が自ら前後のビルド結果を比較して差分を計算する必要がある．またビルド間の差分は，STEP1 でビルドキャッシュ効果の有無によるビルド時間の変動を把握する場合にも用いるため，最適化作業に欠かせない情報であるといえる．

3.　提 案 手 法

3.1　概　　要

2.3 節で述べた課題解決のため，提案ツールの機能要件として以下の 4 点を定めた．

R1. サイズが大きいレイヤを特定できる
R2. ビルドキャッシュが利用されていないレイヤと，そのレイヤによって生じたビルド時間の変動を把握できる
R3. Dockerfile の修正によって得られた効果を把握できる
R4. 問題のレイヤに対応する Dockerfile のソースコード行（修正箇所）を特定できる

本研究ではこれらの要件を満たす提案ツールを，コードエディタの拡張機能として試作した．コードエディタには，開発者の間で高いシェアを誇る[2]Visual Studio Code（VS Code）を採用した．以降では提案ツールの機能について順に説明する．

3.2　機能 1：ビルドと解析結果の描画

提案ツールは VS Code の拡張機能として提供される． 図 2 に示すように，VS Code 上に Dockerfile を開いた状態で画面上部の DfileProfiler ボタンをクリックすると，自動で Docker イメージをビルドし，イメージの解析結果を右側のペインに表示する．イメージの解析結果は相対表示画面と差分表示画面で構成されており，画面の切り替えはタブで行う．各画面の詳細は続く 3.3 節と 3.4 節で説明する．

3.3　機能 2：相対表示画面

図 2 の右側のペインに示すように，相対表示画面では，各レイヤのサイズとビルド時間を棒グラフで表現している．グラフの長さからサイズの大きいレイヤが直感的に分かるため，要件 R1 を満たす狙いがある．さらにサイズの大きいレイヤから順に調べ，Dockerfile の最適化効果が大きい順に最適化を実施できる狙いもある．パレートの法則に当てはめると，Docker イメージのサイズを過剰に大きくさせている 8 割の原因がサイズの大きい上位 2 割のレイヤに集中しているといえるためである．パレートの法則に基づく最適化手法はソフトウェア工学において一般的である．Russell [6] はソフトウェアの最適化において，プロファイリングツールの解析から最大のボトルネックを特定し，上位の問題解決のみに集中するというパレートの法則に基づいた手法を提案している．

また，相対表示画面のビルド時間の欄について，ビルドキャッ

(注1)：https://github.com/wagoodman/dive.git

(注2)：https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment

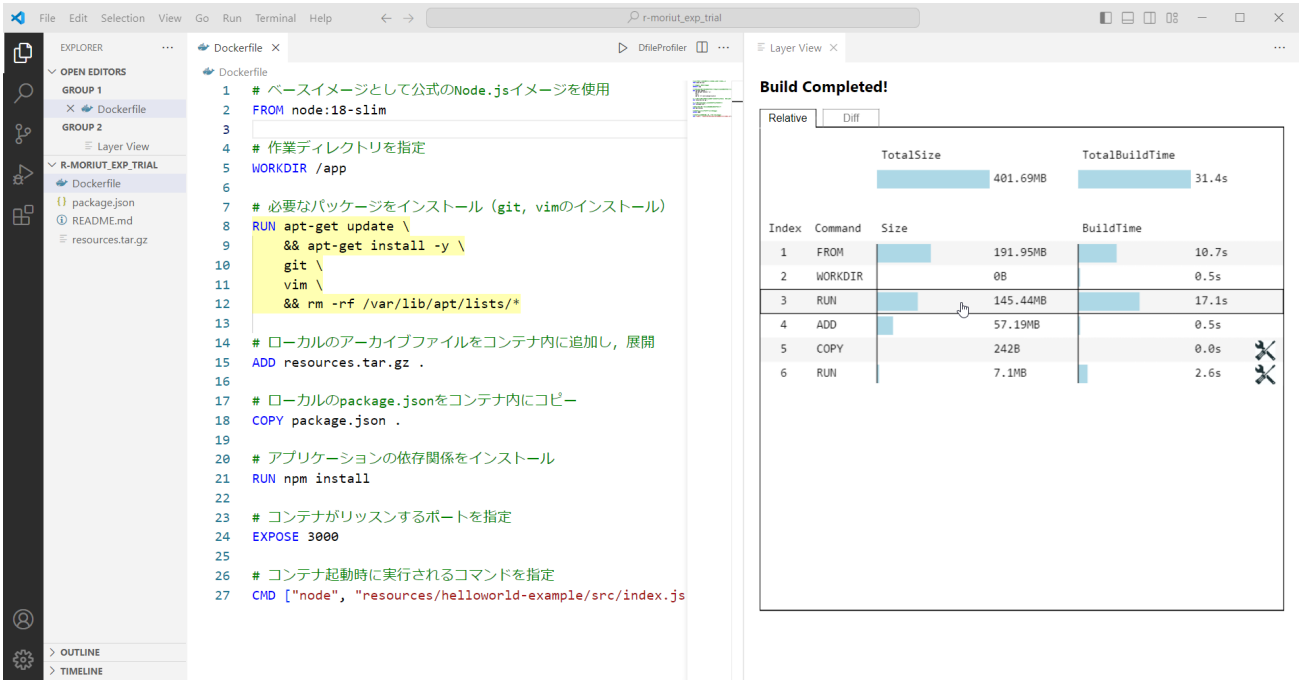

図 2　提案ツールの全体像

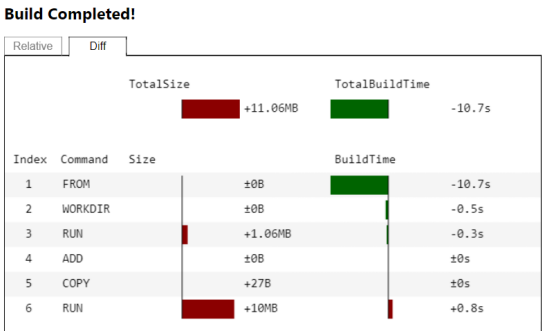

図 3　差分表示画面

シュが利用されているレイヤには，ビルド時間の値のかわりに“CACHED”という文字が表示される設計にした．この設計によって，どのレイヤからビルドキャッシュを利用できなくなるかが一目で判断できるため，要件 R2 の「ビルドキャッシュが利用されていないレイヤを把握できる」という要素を満たす狙いがある．

3.4　機能 3：差分表示画面

図 3 に示すように差分表示画面では，各レイヤのサイズとビルド時間について，1 つ前のビルド結果との差分を棒グラフで表現している．直前のビルド結果と比べて増加している場合は赤色のグラフ，減少している場合は緑色のグラフとなる．グラフの色や長さ，向きなどでどのレイヤのサイズとビルド時間がどれほど増減したかが直感的に分かるため，要件 R2 の「ビルド時間の変動を把握できる」という要素と，要件 R3 を満たす狙いがある．

3.5　機能 4：レイヤと Dockerfile のマッピング

前述の 図 2 に示すように，解析結果のレイヤにマウスオーバーすると，Dockerfile の対応する行がハイライトされる．また，レイヤをクリックすると Dockerfile の対応する行にジャンプする．このようなレイヤと Dockerfile のマッピング機能によって，レイヤと Dockerfile の各行の対応関係が把握しやすくなるため，要件 R4 を満たす狙いがある．

3.6　機能 5：再構築対象のレイヤの通知

Dockerfile や外部の依存ファイルを編集すると，ビルドキャッシュが利用できなくなるレイヤに再構築を示すアイコンが表示される．アイコンの表示と非表示は編集の度にリアルタイムで反映される．たとえば前述の 図 2 に示すように，package.json の中身を編集すると，解析結果における COPY コマンド以降のレイヤにアイコンが表示される．このような再構築対象のレイヤの通知機能によって，どのレイヤからビルドキャッシュを利用できなくなるかがビルド前に確認しやすくなるため，要件 R2 の「ビルドキャッシュが利用されていないレイヤを把握できる」という要素を満たす狙いがある．

4.　実　　験

4.1　概　　要

提案ツールの有効性を評価するために被験者実験を実施した．実験では，Dockerfile の最適化タスクを被験者に取り組ませる．被験者は提案ツール使用の有無で 2 グループに分けた．評価指標には正解率と事後アンケートの回答結果を用いた．

4.2　被験者とグループ分け

本実験の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 3 名の計 12 名である．

まず，被験者全員に対して事前アンケートを行った．アンケートでは，ドレイファスモデル [7] を元に Docker の習熟度を「初心者」「中級者」「上級者」「熟練者」「達人」の 5 段階に

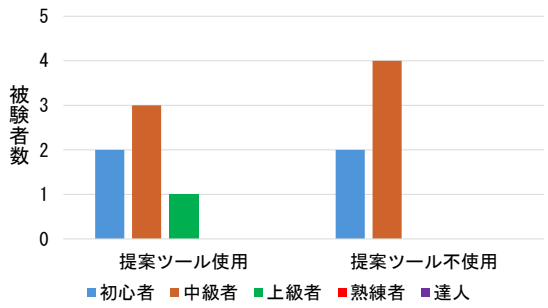

図 4　グループ別の Docker の習熟度の分布

区別し，被験者自身が最も当てはまると思う習熟度を選択する形式にした．

次に，事前アンケートの回答結果に基づいて，被験者を「提案ツール使用」と「提案ツール不使用」の 2 グループに分けた．グループ別の Docker の習熟度の分布を 図 4 に示す．グループ間での能力の偏りを防ぐため，Docker の習熟度の分布がなるべく同じになるよう振り分けた．

なお，「提案ツール使用」グループの被験者に対しては，本実験の前にツールの使い方を 30 分ほど学習させた．被験者間で提案ツールの使用に慣れるまでの差を排除し，提案ツールの有効性をより正確に評価できるようにするためである．学習用に用意した簡潔な Dockerfile を対象に，3.2 節〜3.6 節で述べた各機能を提案ツールの操作を通して確認させた．

4.3　Docker の事前学習

被験者全員に対して，本実験の前に 60 分ほど Docker の事前学習を行った．被験者の Docker への習熟度が提案ツール使用者の想定レベルを満たすようにすることで，より公正な評価ができるようにするためである．前述の 図 4 に示すように，被験者の 30% は初心者で Docker を全く知らない状態である．対して，提案ツールの想定レベルは中級者以上である．具体的な想定レベルとして，Docker の基本的な概念を理解しており，Docker の公式ドキュメント[3]を参考に Dockerfile の作成や Docker イメージのビルドができる状態が望ましい．このような想定レベルを踏まえ，Docker に関する基本的な用語と，実験タスクに取り組むために必要な知識を学習内容に取り入れた．

4.4　タ　ス　ク

4.4.1　概　　要

実験タスクは Dockerfile の最適化である．タスクの設計においては，Dockerfile を用いた Web フロントエンドの開発を想定した．Web 開発は Docker のユースケースでよく見られる．Haque ら [8] は Stack Overflow 上の Docker に関連する投稿を調査し，Web サーバと Web フレームワークの設定や WebDriver の使い方など，Web 開発に関連する質問が多数見られたと報告している．このように Web 開発をシナリオに設定することで，提案ツールが実用的な場面においても有効であることを確認する．なお，タスクは 2 種類に分けられる．Docker イメージのサイズ削減を目的とした最適化タスクと，ビルド時間の削減を目的とした最適化タスクである．

4.4.2　イメージサイズ削減タスク

本タスクは「Q1：問題点の発見」「Q2：問題点の分析」「Q3：問題点の修正」「Q4：修正効果の把握」の 4 つの設問で構成される．各設問の内容を 表 1 に示す．題材の Dockerfile には，パッケージリストの削除忘れや不要なファイルのコピーなどイメージサイズを大きくさせる問題点が複数含まれている．被験者には問題点それぞれについて，30 分間で Q1〜Q4 を繰り返し実施させた．

4.4.3　ビルド時間削減タスク

本タスクは 4.4.2 節同様，「Q1：問題点の発見」「Q2：問題点の分析」「Q3：問題点の修正」「Q4：修正効果の把握」の 4 つの設問で構成される．各設問の内容を 表 2 に示す．題材の Dockerfile には，ビルドキャッシュの不十分な活用といったビルド時間を増加させる問題点が 1 つ含まれている．被験者にはその問題点について，15 分間で Q1〜Q4 を実施させた．

4.5　評 価 指 標

実験の評価指標には，正解率と事後アンケートの回答結果を用いる．

正解率とは全体の設問数に対して正解した設問数の割合である．被験者ごとにタスクの各設問の解答を採点して正解率を算出し，「提案ツール使用」グループと「提案ツール不使用」グループのそれぞれで平均正解率を算出した．

また，「提案ツール使用」グループの被験者 6 名に対して事後アンケートを行った．アンケートはリッカート尺度 [9] での 5 段階評価で回答する質問と，自由記述で回答する質問で構成される．5 段階評価で回答する質問では，機能要件で定めた問題点の発見，修正効果の把握，レイヤと Dockerfile のマッピングに提案ツールが役立ったか尋ねたほか，直感的な操作や動作の軽さなど，提案ツールが満たすべき非機能要件についてどのように感じたか尋ねた．自由記述で回答する質問では，提案ツールの良かった点や改善すべき点を尋ねた．

5.　結果と考察

5.1　イメージサイズ削減タスクの平均正解率

イメージサイズ削減タスクの平均正解率を 図 5 に示す．図を見ると，いずれの設問も提案ツールを使用した方が平均正解率がおよそ 3〜13% 高くなっている．特に Q4 の平均正解率については，およそ 2 倍の改善が見られた．

「Q1：問題点の発見」と「Q4：修正効果の把握」の平均正解率が向上した理由としては，Q1 では提案ツールの相対表示画面の機能が，Q4 では差分表示画面の機能が有効に働いたためと考えられる．さらに Q4 の解答を分析すると「提案ツール不使用」グループの被験者からは，適切な差分値からかけ離れた明らかに計算ミスの可能性が高い解答が散見された．このような計算ミスを防ぐ提案ツールの強みが平均正解率の向上に貢献したともいえる．

また，「Q2：問題点の分析」と「Q3：問題点の修正」の平均正解率が向上した理由としては，Q1 と Q4 に解答する効率が良くなった結果，副次的に Q2 と Q3 に取り組む時間が増えた

(注3)：https://docs.docker.com/

表 1　イメージサイズ削減タスクにおける各設問の内容

項目	内容
Q1：問題点の発見	サイズが大きいレイヤと，そのサイズの値を答えてください．
Q2：問題点の分析	Q1 のレイヤーは，なぜサイズが大きいのでしょうか．その要因を答えてください．
Q3：問題点の修正	Q2 の要因について，改善策を考えて Dockerfile を修正してください． 修正が完了したら，どのように修正したか答えてください．
Q4：修正効果の把握	Q3 の修正によって，効果がどれくらい得られたか答えてください．

表 2　ビルド時間削減タスクにおける各設問の内容

項目	内容
Q1：問題点の発見	index.html の中身を資料記載の内容（ここでは省略）に丸ごと差し替えてください． 差し替え後 Docker イメージをビルドしてビルド時間がどれだけ増加したか答えてください．
Q2：問題点の分析	Q1 のファイルの編集でなぜビルド時間が増加したのでしょうか．その要因を答えてください．
Q3：問題点の修正	Q2 の要因について，改善策を考えて Dockerfile を修正してください． なお，今後 index.html を含めた Web サイトのリソースを更新する頻度は極めて高いと想定してよいです． 修正が完了したら，どのように修正したか答えてください．
Q4：修正効果の把握	index.html について資料記載の内容（ここでは省略）に変更してください． 変更後 Docker イメージをビルドして，Q3 の修正によって効果がどれくらい得られたか答えてください．

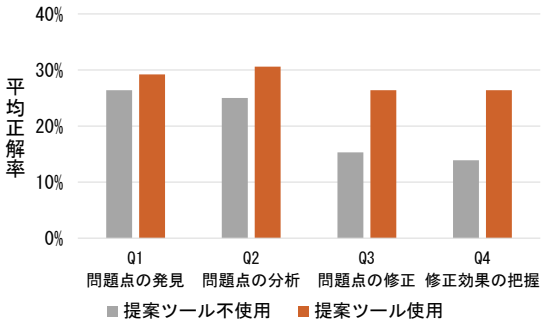

図 5　イメージサイズ削減タスクの平均正解率

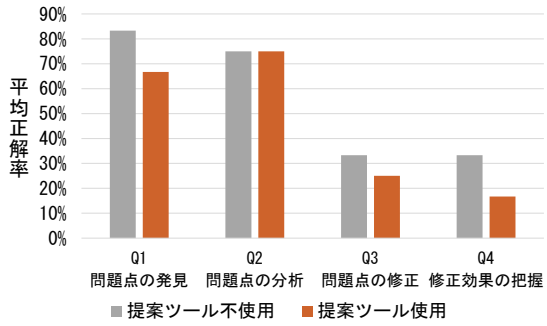

図 6　ビルド時間削減タスクの平均正解率

ためと考えられる．

一方で，提案ツール使用の有無に関わらず平均正解率は最高でおよそ 25〜30% であり，全体を通して低い結果となっている．これは最適化で実施する一連の流れに被験者が慣れておらず，被験者のレベルに対してタスクの難易度が上がってしまった点が原因だと考えられる．

5.2　ビルド時間削減タスクの平均正解率

ビルド時間削減タスクの平均正解率を 図 6 に示す．図を見ると，Q1 と Q3，Q4 は提案ツールを使用した方が平均正解率がおよそ 8〜16% 低くなっている．

「Q1：問題点の発見」と「Q4：修正効果の把握」の平均正解率が向上しなかった理由としては，提案ツールの差分表示画面の機能が有効に働かなかったためと考えられる。要因として，ビルド時間の履歴を提案ツール上で遡って確認できなかった点が大きい．本タスクではビルドキャッシュ効果の有無によるビルド時間の変動を観察しなければならない．提案ツールではビルド結果の差分は直近との比較に限られているが，ターミナルではウィンドウを閉じない限りビルド結果を遡って確認できる．外部ファイルの編集ミスや Dockerfile の修正ミスによってビルドの試行錯誤を繰り返してしまうと，提案ツールではビルド時間の変遷を追えなくなるという欠点があった．

また，「Q2：問題点の分析」と「Q3：問題点の修正」の平均正解率が向上しなかった理由としては，Q1 に解答する効率が悪くなった結果，副次的に Q2 と Q3 に取り組む時間が減ったためと考えられる．

5.3　事後アンケート

5 段階評価での回答結果を 図 7 に示す．図の上記 5 つの項目に対して，いずれも被験者の 80% 以上が肯定的な回答を示した．また自由記述での回答では，提案ツールの良かった点について 4 名の被験者が差分表示画面の機能を挙げていた．他にも 2 名の被験者が相対表示画面の機能を，1 名の被験者がレイヤと Dockerfile のマッピング機能と再構築対象のレイヤの通知機能を挙げていた．一方で 3 名の被験者が「直近との差分だけではなく，ビルド結果の差分を遡って閲覧できる機能がほしい」といった意見を挙げており，5.2 節で述べたビルド時間削減タスクで「提案ツール使用」グループの平均正解率が低下した要因がここからも確認できる．

他方で 図 7 を見ると，「動作は軽く遅延もなかった」という項目に対して被験者のおよそ 30% が否定的な回答をしている．自由記述での回答では，改善すべき点について 1 名の被験者

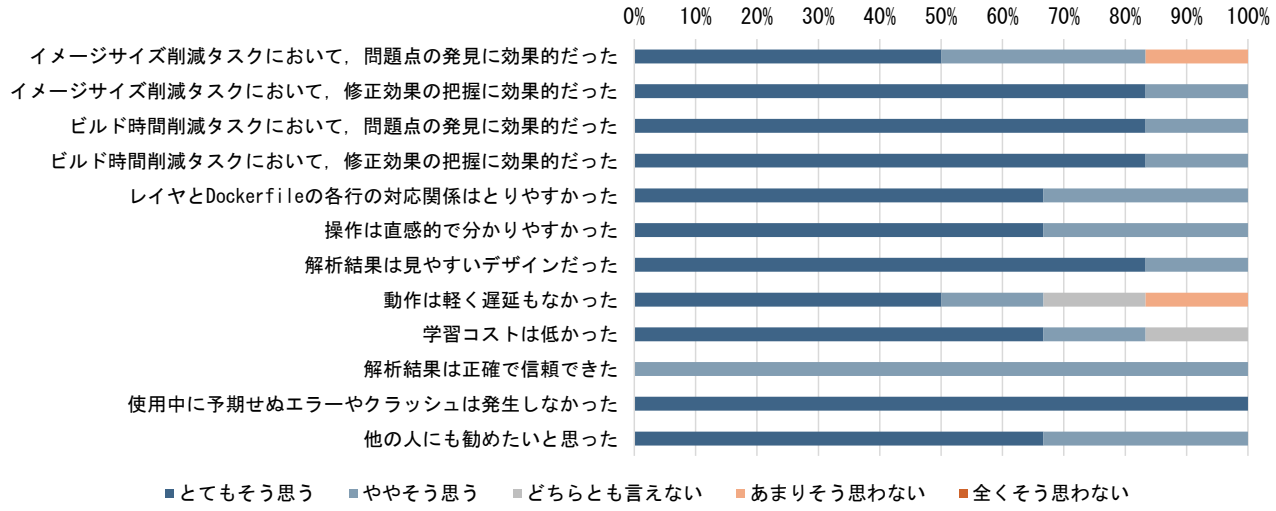

図 7　提案ツールに関する事後アンケートの 5 段階評価での回答結果

が「描画処理が長すぎる」といった意見を挙げており，被験者の環境によって提案ツールの処理時間に大きな差異が出てしまったと考えられる．

6. おわりに

本研究では Docker イメージの品質改善を目的とした Dockerfile 最適化支援ツールを試作した．さらに，提案ツールの有効性を評価するために被験者実験を実施した．実験の結果，提案ツールはイメージサイズ削減タスクに対して正確性と効率性の向上に効果があることを確認した．また事後アンケートの回答結果から，提案ツールで実装した全ての機能は Dockerfile の最適化に有効であることを確認した．

今後の課題として，提案ツールの差分表示画面の機能を修正する必要がある．過去のビルド結果から任意の 2 点を選択して差分を確認できる機能に改善することで，イメージサイズ削減タスクのみならず，ビルド時間削減タスクに対しても正確性と効率性の向上が期待できる．また非機能要件である処理効率も改善が必要であり，動作環境に依存しない実装方法の見直しが求められる．

さらに，提案ツールの発展的な改善を検討している．本研究では，最適化作業の 4 ステップのうち「STEP1：問題点の発見」と「STEP4：修正効果の把握」に着目していた．今後は「STEP2：問題点の分析」と「STEP3：問題点の修正」に着目し，問題点の要因と修正案の提示機能を追加すべきである．たとえばレイヤごとの依存パッケージを特定し，どのレイヤにも依存しないパッケージを提示して不要なパッケージの削除を支援する機能や，レイヤの変更頻度をもとに，コマンドの順序を最適化してビルドキャッシュの活用を支援する機能などが挙げられる．これらの改善案ではビルド時の動的情報が欠かせない．STEP2 と STEP3 の支援を目的とした研究は存在するが，ソースコード上の静的情報から導き出される要因や修正案の提示に限られている [10] [11]．STEP2 と STEP3 で動的情報を有効活用し，STEP1〜STEP4 を包括的に支援する提案ツールに改善することで，ツールの価値の大幅な向上が期待できる．

謝辞　本研究の一部は，JSPS 科研費（JP24H00692, JP21H04877，JP21K18302）による助成を受けた．

文　献

[1] Y. Wu, Y. Zhang, K. Xu, T. Wang, and H. Wang, “Understanding and predicting docker build duration: An empirical study of containerized workflow of OSS projects,” In Proceedings of International Conference on Automated Software Engineering, pp.1–13, 2022.

[2] T. Gkamas, V. Karaiskos, and S. Kontogiannis, “Performance evaluation of distributed database strategies using docker as a service for industrial IoT data: Application to industry 4.0,” Journal on Information, vol.13, no.4, p.190, 2022.

[3] J. Qian, Y. Wang, X. Wang, P. Zhang, and X. Wang, “Load balancing scheduling mechanism for openstack and docker integration,” Journal on Cloud Computing, vol.12, no.1, p.67, 2023.

[4] C. Lin, S. Nadi, and H. Khazaei, “A large-scale data set and an empirical study of docker images hosted on docker hub,” In Proceedings of International Conference on Software Maintenance and Evolution, pp.371–381, 2020.

[5] E. Ksontini, M. Kessentini, T. doN. Ferreira, and F. Hassan, “Refactorings and technical debt in docker projects: An empirical study,” In Proceedings of International Conference on Automated Software Engineering, pp.781–791, 2021.

[6] S. Russell, T.D. Bennett, and D. Ghosh, “Software engineering principles to improve quality and performance of R software,” Journal on PeerJ Computer Science, vol.5, no.1, p.e175, 2019.

[7] H.L. Dreyfus and S.E. Dreyfus, “A five-stage model of the mental activities involved in directed skill acquisition,” Technical Report ORC-80-2, Operations Research Center, University of California, Berkeley, 1980.

[8] M.U. Haque, L.H. Iwaya, and M.A. Babar, “Challenges in docker development: A large-scale study using stack overflow,” In Proceedings of International Symposium on Empirical Software Engineering and Measurement, pp.1–11, 2020.

[9] R. Likert, “A technique for the measurement of attitudes,” Journal on Archives of Psychology, vol.22, no.140, pp.5–55, 1932.

[10] J. Henkel, C. Bird, S.K. Lahiri, and T. Reps, “Learning from, understanding, and supporting devops artifacts for docker,” In Proceedings of International Conference on Software Engineering, pp.38–49, 2020.

[11] T. Durieux, “Empirical study of the docker smells impact on the image size,” In Proceedings of International Conference on Software Engineering, pp.2568–2579, 2024.