シングル GPU による日本語コード LLM の構築

相馬菜生 [1]　小原百々雅 [1]　小原有以 [2]　高橋舞衣 [1]　佐藤美唯 [1]　倉光君郎 [2]
[1] 日本女子大学大学院 理学研究科　[2] 日本女子大学 理学部
{m1916045sn,m1816019om}@ug.jwu.ac.jp　kuramitsuk@fc.jwu.ac.jp

概要

昨今，大規模言語モデル（LLM）の研究と開発は顕著な進歩を遂げている．LLM は自然言語処理分野において優れた成果を上げると同時に，プログラミングを含む多岐にわたるタスクで高いパフォーマンスを示している．しかし，大学や個人研究者などリソースが限られた環境下では，これらのモデルの扱いが容易ではない．本研究では，日本語コード LLM と HumanEval データセットを対象としたスケールアップ基盤の確立を目指す．この取り組みにより，限られたリソースを有する研究者たちも LLM の研究開発に参加し，新たなアイディアや手法を容易かつ効果的に検証する機会を得ることが期待される．

1　はじめに

2023 年は，LLM の普及と活用が急速に拡大した一年であった．LLM の驚異的な可能性に世界中の関心が集まり，国内外の研究機関において LLM の開発も活発化した．この結果，LLM の開発がコーパスの整備から評価尺度の設定に至るまで，ビッグサイエンスの領域に入り込んでいるということが明らかとなった．今後，高品質な LLM の開発には，裾野の広い研究者や開発者の参画が不可欠である．

しかし，LLM の開発には膨大な計算機リソースが必要であるため，大学研究室レベルの組織では気軽に挑戦できるテーマではない [1]．最近では，パラメータ数の少ない小規模な LLM の開発 [2] にも関心が集まるが，それでもパラメータ数は 1B を超え，最低でも 8 台以上の最先端 GPU を必要とする．

本研究の目的は，小規模コード LLM への開発経験を報告し，研究室レベルのリソースを用いた LLM の研究開発をスケールアップする道筋を示すことである．本研究では，シングル GPU(V100, A100) を研究室レベルのリソースと想定している．

小規模 LLM の開発における主要な課題は，限られたパラメータ数により，有意な評価を得ることが困難であることにある．著者らの経験でも，期待される能力が示現しない状況下で，いつまで学習を継続すべきかの判断がしばしば課題となる．このような状況は，研究基盤として新たなアイディアを比較し，分析することを難しくしている．

本研究の貢献は，シングル GPU 学習のみで開発可能な小規模 LLM を対象として，HumanEval データセット [3] の正解率を評価できる小規模 LLM の開発経験を示したことである．HumanEval は，Copilot など LLM の実運用サービスの開発につながる実体のある評価尺度である．我々は，スケールアップした高性能な LLM 開発への見通しを立てやすくするため，複数スケールの小規模 LLM(0.06B, 0.13B, 0.3B, 0.6B, 1.3B) の性能の比較評価に取り組んだ．また，小規模 LLM の能力を解像度高く統計的に比較するため，HumanEval ベンチマークを小規模モデル用に改良した HumanEval-Easy37 を開発した．これにより，前処理や学習手順などの違いをより明確に把握できるようになった．

2　コード LLM

本論文では，LLM のコード生成能力に着目して日本語の小規模 LLM の開発を目指す．

2.1　日本語対応のコード LLM

コード LLM とは，コード生成のために特別に学習された LLM のことである．最近は，事前学習データセットにコードを含めることが一般的になっているが，LLM であれば高いコード生成能力を持つものではない [3, 4]．高いコード生成能力を得るためには，事前学習データセットや微調整に工夫が必要である [5, 6, 2]．なお，モデルアーキテクチャ，必要な計算機資源，スケーリング則 [7, 8] は，基本的に汎用的な LLM と同じである．そのため，コード LLM は，特定のドメインに特化した LLM と考えることができる．

我々の最終的な目的は，日本語に対応したコード LLM を可能な限り小規模に開発することである．これは相反する目標ではなく，LLM のドメインを絞ることにより，より少ない訓練データや学習時間でコンパクトな LLM[9] の実現が可能だからである．一方，日本語で書かれたコード関係のテキストが圧倒的に少なく，新しい工夫が必要な点が課題として存在する．事実，日本語に対応したコード LLM の開発はまだ盛んではない．

2.2 HumanEval

コード LLM において，代表的な評価項目は自然言語記述からのコード生成能力である．しかし，生成されたコードが正しいかどうかを判定するのは，用意ではない．これまで，長らく機械翻訳の評価尺度である BLEU[10], コード生成向けに改良した CodeBLUE[11], 編集距離に基づくレーベンシュタイン類似度 [12] などが評価尺度として使われてきた．しかし，コードは少しの字句の違いで解釈が大きく変わるため，これらは必ずしもコード LLM の評価に適していないことも明らかだった．ソフトウェアテストに基づく正解率 (Computational Accuracy) [13] も提案されていたが，初期のコード LLM[14, 15] では実行可能なコードを出力すること自体が容易ではなかった．強力なコード生成能力は，GPT-3 に基づく Codex の登場まで実現されなかった．

HumanEval[3] は，OpenAI 社の Codex の発表時に採用された，ソフトウェアテストに基づくコード生成能力の評価尺度である．164 個のプログラミング問題からなるデータセットであり，回答に数学や物理学，英語などの知識を要する問題も含まれている．すなわち，自然言語の意味を適切に理解し，ユニットテストをクリアするためのコードを生成する能力が必要とされる．重要なのは，字句的に全く異なるコードであっても，その機能的な意味が同一であれば，正解として評価される点である．HumanEval によるコード生成能力の評価は，Copilot などの実用的な LLM 応用の実現において，重要な役割を果たしている [16, 17].

3 コーパスの前処理

本節では，我々が小規模 LLM 向けに前処理を実施した訓練データセットについて述べる．

表 1 準備したコーパス量

データセット	トークン量
Python コード (docstring, コメント英文/日本文)	4.4 B
Python コード (docstring, コメントなし)	2.0 B
Github Markdown 文書 (英文, Python 関連)	2.6 B
mC4 日本語コーパス (クリーニング済)	28.3 B

3.1 基本方針

本研究で開発するコード LLM は，対応言語を日本語，英語，Python コードに絞り，学習を行う．また，バランスの取れた学習を実現するため，Huggingface Hub に公開されている複数のデータセットを活用する．

トークナイザに関しては，既存の LLM-jp トークナイザ (v2.1) を採用する．これにより，我々が開発した小規模 LLM の，LLM-jp プロジェクトにおける投機予測や連邦モデルなどへの活用が容易になる．LLM-jp トークナイザは，日本語文，英文，コードから構築された語彙モデルをマージすることで構築されている．

データセットの前処理は，大規模な LLM と同様のアプローチを採用する．ただし，小規模 LLM の場合，パラメータ数が少ないため，以下のような予測しにくい文字列の学習が学習効率に不利益を及ぼす可能性がある．この問題に対処するため，正規表現を用いた積極的な文字列の置換及び除去を行い，効率的な学習プロセスを確保した．

- 日付 (2024-01-12, 2024/01/12 など) 時刻
- 住所，電話番号
- メールアドレス，アカウント名など
- URL，長いファイルパス
- HTML タグ，文字装飾，ルビ，カッコ書き
- UUID，ハッシュ値，BASE64 データ
- 数値データの羅列

3.2 コード・英文コーパス

一般的に，コードデータセットは BigCode プロジェクトが公開した The Stack[18] を採用することが多いが，我々は StarCoder[19] 開発用の StarCoderData をベースに整備した．StarCoderData の特徴は，The Stack に特定の前処理を加えることで，個人情報や URL などが置き換えられている点である．我々は，StarCoderData のデータセットから，ファイル拡張子に基づいて，Python スクリプトと Markdown 文書のみ抽出し，以下の前処理を追加で行った．

表 2　既存の小規模コード LLM パラメータ一覧

モデル	Parameters	d_model	n_dims	n_layers	n_heads	intermediate_size
pythia-410m	410M	1,024	64	24	16	4,096
phi-1-small	350M	1,024	64	20	16	4,096
phi-1	1.3B	2,048	64	24	32	8,192

ソースコードに関しては，コメントに対して前処理をする．まず，定型的なライセンス文書やコードのコメントアウトは取り除き，英文と日本語文のみ取り出す．英文は，コードとコメント文の比率が10-50%程度含まれているものを選んだ．日本語のコメント文は，日本語コメントがある Python ファイルが全体の 0.3%程度であったため，全て採用した．また，全てのコードにコメントが存在するのは不自然であるため，コメントなしのコードもデータセットに含めた．

Markdown 文書は，キーワードで Python 関係の文書に限定し，英文か日本語文が含まれるセクションを抽出した．ただし，英文は頻出英単語の含有率が低いときは除外した．Markdown に埋め込まれたコードやデータは，前処理の対象から外したが，2,048 文字を上限で切り詰めた．

3.3　日本語コーパス

英文は，コードのコメントや Markdown 文書から収集した．一方，日本語文は量が不十分であったため，多言語データセット mC4[20] の日本語全文を用いた．mC4 は，Web クローラで収集した文章であるため，提示版のページナビゲーションや広告などの LLM の学習にふさわしくない情報が大量に含まれている．そのため，正規表現でできるだけ取り除いた．さらに，HojiChar[21] の有害語ブロックリストを用いて，有害語が 3 種類以上含まれる文書はフィルタした．日本語の品質は，助詞が含まれている比率を概算して，助詞が少ない場合は低品質としてフィルタの対象とした．加えて，日付，個人情報，URL，UUID などは，訓練に不向きな文字列として，積極的に置き換えを行った．その結果，日本語データセットは，最終的に 5 分の 1 ほどに減った．

我々が用意したデータセットは，表 1 に示す通りである．

4　LLM の開発

4.1　事前学習

我々は，異なる 5 段階のパラメータ数の Llama2 をスクラッチ生成し，3 節で用意したコーパスを用いて Causal Language Model (CLM) 事前学習することで，小規模コード LLM の構築を行う．ただし，本研究で構築する小規模コード LLM のパラメータ数は，0.06B, 0.13B, 0.3B, 0.6B, 1.3B の 5 種類である．既存の小規模モデルのハイパーパラメータを参考にして (表 2)，それぞれ layer 数や head 数などのハイパーパラメータを調整することで決定した．本研究で調整した各モデルのハイパーパラメータを表 3 に示す．

表 3　小規模コード LLM パラメータ一覧

	Parameters (millions)	n_dims	n_layers	n_heads	intermediate_size
0.06B	65.2M	64	6	8	768
0.13B	138.2M	64	16	10	1,536
0.3B	321.7M	64	16	16	3,072
0.6B	647.1M	64	20	22	4,096
1.3B	1,287.9M	64	24	24	8,192

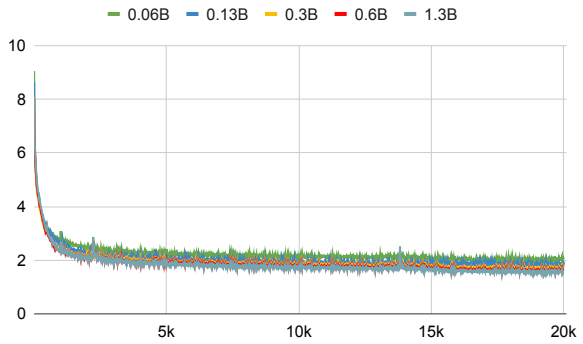

図 1　損失関数 (train/loss)

モデル構築の学習環境には，産業総合研究所の AI 橋渡しクラウド (ABCI) のシングル GPU(A100) を使用した．本研究では，シングル GPU で 1epoch 事前学習した．

学習率スケジューラを constant に設定し，学習率 5e-4 で学習を開始し，5kstep(0.13B のみ 10kstep) 毎にチェックポイントとし，構築を進めた．必要に応じて学習率を 5e-5 まで下げて学習を行った．

パラメータ毎の小規模コード LLM 訓練時における損失関数は，図 1 の通りである．

KOGITUNE は，我々が独自に開発した分散学習基盤である．テキスト前処理などの時間がかかる処理は別の計算機で行い，訓練データはテンソル化した状態で非同期のネットワーク配信し，ほとんど CPU 負荷をかけずにテンソルをメモリにロードすることができる．そのため，KOGITUNE を用いることで，ほぼ 100%GPU 利用率を達成できた．

4.2 HumanEval-Easy37

本節では，HumanEval ベンチマークを小規模モデルの評価用に開発したベンチマーク HumanEval-Easy37 について述べる．

HumanEval-Easy37 は，HumanEval のデータセットのうち，簡単な問題を中心に集めて開発したベンチマークである．問題抽出の流れとしては，まず，異なるモデルを 7 つ準備し，HumanEval を用いて結果を算出した．そして，算出した結果をもとに 164 件のうちモデルの正答数が多い簡単な問題 37 件を抽出し，データセットとした．

比較的難易度の低い問題に絞って，試行回数を増やしてテストしやすくすることで，小規模モデルの能力に合わせた分析がしやすくなることが期待される．

4.3 スケール評価

まず，4.1 節で用意した 5 つのスケールの小規模コード LLM の性能比較を行う．小規模コード LLM の HumanEval，HumanEval-Easy37 での評価結果をそれぞれ図 2, 3 に示す．ただし本評価では，HumanEval，HumanEval-Easy37 ともに pass@1(HumanEval では n=1，HumanEval-Easy37 では n=10) で算出した結果である．

多くのモデルで，step 数の増加に伴いスコア向上の傾向が見られた．しかし， 図 2 より 0.6B の step 数が 25k の段階で，HumanEval の値が低下しているように，一部性能向上が止まったり，低下するケースも見られた．この現象は，learning_rate の調整不足による過学習の可能性が疑われる．本研究で構築した LLM の中で最もスケールの小さい 0.06B は，167 問中 1 問のみ正しいコードを生成できることが確認された．更に，0.06B では，step 数の増加が性能向上に寄与していない．他のモデルと比較しても 0.06B は効果的な学習ができていないと推測される．一方で，0.13B の小規模コード LLM は，基本的に step 数を増やすほど正解率が向上している．これらにより，本研究で構築した小規模なコード LLM のうち， 0.13B が小規模且つ有望な LLM であることが明らかとなった．また，HumanEval-Easy37 の評価結果 (図 3) と HumanEval の評価結果 (図 2) を比較すると大きく違う点はないが，図 3 の方が step 数が増えるとスコアが上がっていく傾向がよりはっきり現れている．

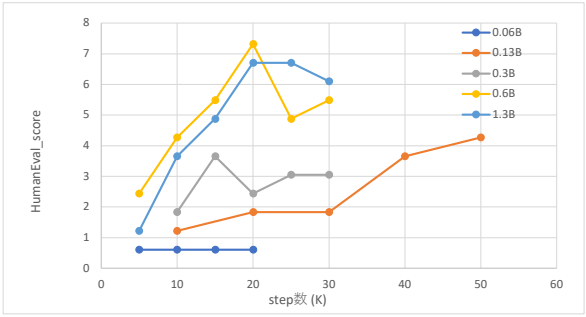

図 2 HumanEval での評価 Pass@1(n=1)

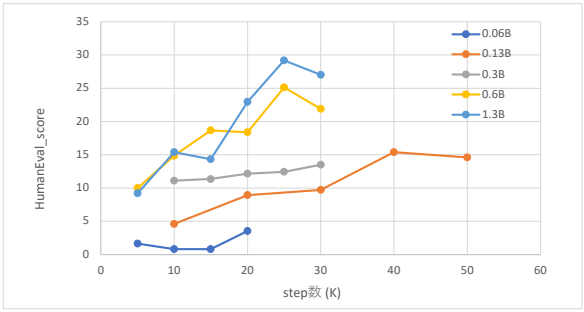

図 3 HumanEval-Easy37 での評価 Pass@1(n=10)

以上のスケール評価より，それぞれのスケールのモデルにおいて以下の 5 点が明らかとなった．

0.06B：ほとんど性能評価の向上が乏しい

0.13B：安定した性能向上が観察された

0.3, 0.6B：シングル GPU で学習可能であったが，learning_rate の調整が必要である

1.3B：既存の小規模 LLM モデルで採用される [2] が，シングル GPU で最後まで学習し切るのは難しい

5 まとめ

我々の最終的な目標は，できるだけコンパクトで，性能の高い日本語対応コード LLM を構築することである．そのため，本論文では，スケールアップ基盤の確立を目指して小規模なコード LLM の構築を行なった．シングル GPU で学習可能な 5 段階のスケールの小規模コード LLM を構築し比較評価を行った結果，研究室レベルの計算機資源による小規模 LLM を，スケーリング則に基づいてより大規模な LLM 開発の性能予測に活用できるおおよその見通しがたった．今後は学習率やモデルパラメータの調整などを行い，さらに詳しくスケールアップに向けた検証を行いたい．更に，ファインチューニングの性能などを評価し，我々の目標とするコード LLM の構築に繋げていきたい．その際に，今回得られたスケールアップに関する知見が役立つことが期待できる．

謝辞

本研究は JSPS 科研費 JP23K11374 の助成を受けた．2023 年度 国立情報学研究所 公募型共同研究「大規模言語モデルの効率良い学習のための訓練データ配信基盤の研究」の一部として実施された．

参考文献

[1] Jonas Geiping and Tom Goldstein. Cramming: Training a language model on a single gpu in one day, 2022.

[2] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. **arXiv preprint arXiv:2306.11644**, 2023.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. **arXiv preprint arXiv:2107.03374**, 2021.

[4] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, and Wenhan Xiong et al. Code llama: Open foundation models for code, 2023.

[5] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle, 2022.

[6] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis, 2023.

[7] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[8] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[9] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023.

[10] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In **Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics**, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[11] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. **CoRR**, Vol. abs/2009.10297, , 2020.

[12] Li Yujian and Liu Bo. A normalized levenshtein distance metric. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 29, pp. 1091–1095, 2007.

[13] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In **Proceedings of the 34th International Conference on Neural Information Processing Systems**, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[14] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, 2021.

[15] Yuka Akinobu, Momoka Obara, Teruno Kajiura, Shiho Takano, Miyu Tamura, Mayu Tomioka, and Kimio Kuramitsu. Is neural machine translation approach accurate enough for coding assistance? In **Proceedings of the 1st ACM SIGPLAN International Workshop on Beyond Code: No Code**, BCNC 2021, p. 23–28, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. Github copilot ai pair programmer: Asset or liability?, 2023.

[17] Nhan Nguyen and Sarah Nadi. An empirical evaluation of github copilot's code suggestions. In **2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)**, pp. 1–5, 2022.

[18] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code, 2022.

[19] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, and Jenny et al Chim. Starcoder: may the source be with you!, 2023.

[20] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, **Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**, pp. 483–498, Online, June 2021. Association for Computational Linguistics.

[21] 新里顕大, 清野瞬, 高瀬翔, 小林滉河, 佐藤敏紀. Hojichar: テキスト処理パイプライン. NLP 若手の会 (YANS) 第 18 回シンポジウム, 2023.

5.1 付録 (Appendix)

以下に，4.3 節の図 2,3 に示した小規模コード LLM の HumanEval,HumanEval-Easy37 に関する評価結果を，PPL の値と共に, より詳細な表としてまとめる．

表 4 HumanEval，HumanEval-Easy37 での評価 pass@1

Parameters (billions)	Steps	Hours	Tokens	PPL	HumanEval (pass@1，n=1)	HumanEval-Easy37 (pass@1，n=10)
0.06b	5k	4.8	2.6B	15.8	0.6	1.6
0.06b	10k	9.7	5.2B	10.0	0.6	0.8
0.06b	15k	14.6	5.2B	8.57	0.6	0.8
0.06b	20k	19.5	10.5B	7.81	0.6	3.5
0.13b	10k	38.7	10.2B	6.95	1.2	4.6
0.13b	20k	77.4	20.4B	6.61	1.8	8.9
0.13b	30k	116.1	30.6B	6.32	1.8	9.7
0.13b	40k	154.8	40.8B	5.79	3.7	15.4
0.13b	50k	192.5	26.2B	5.66	4.3	14.6
0.3b	10k	30.4	5.2B	6.30	1.8	11.0
0.3b	15k	45.6	7.9B	5.82	3.6	11.4
0.3b	20k	60.9	10.5B	5.54	2.4	12.2
0.3b	25k	76.1	13.1B	5.49	3.0	12.4
0.3b	30k	91.5	15.7B	5.33	3.0	13.5
0.6b	5k	28	2.6B	7.65	2.4	10.0
0.6b	10k	56	5.2B	5.94	4.3	14.9
0.6b	15k	84	7.9B	5.50	5.5	18.7
0.6b	20k	112	10.5B	5.13	7.3	18.4
0.6b	25k	140	13.1B	5.13	4.9	25.1
0.6b	30k	168	15.7B	5.15	5.5	21.9
1.3b	5k	48.2	2.6B	6.73	1.2	9.2
1.3b	10k	96.4	10.5B	5.64	3.7	15.4
1.3b	15k	144.6	10.5B	5.50	4.9	14.3
1.3b	20k	192.8	10.5B	5.64	6.7	23.0
1.3b	25k	241.1	10.5B	5.18	6.7	29.2
1.3b	30k	289.3	15.7B	4.67	6.1	27.0