Generating Code Tours Using Locally-Runnable LLMs

Martin Balfroid
martin.balfroid@unamur.be
NADI, University of Namur
Namur, Belgium

Abstract

Onboarding new developers is a considerable overhead, requiring senior developers to provide mentorship and documentation. Code tours—structured guides to key code sections—can streamline this process, yet their manual creation still requires time and effort. This research investigates how to automate the generation of debugging-focused code tours using locally runnable Large Language Models (LLMs), which provide more confidentiality, stability, and reproducibility than cloud-based models. We plan to address three key challenges: (1) selecting relevant code segments, (2) generating clear and context-aware explanations, and (3) automatically evaluating the quality of generated tours. Ultimately, our goal is to reduce the onboarding burden on senior developers by automatically generating persistent, verifiable documentation artifacts to help junior developers navigate and understand a codebase.

CCS Concepts

• **Computing methodologies** → **Natural language generation**; • **Software and its engineering** → **Software development process management**.

Keywords

large language models, code tour, developer onboarding

ACM Reference Format:
Martin Balfroid. 2025. Generating Code Tours Using Locally-Runnable LLMs. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3696630.3731462

1 Introduction

Onboarding new developers into an organization is crucial but challenging; the most recommended strategies are mentoring, debugging, and documenting [8, 10]. Still, it requires significant time and effort from senior developers - up to 30% in the case of the CERN project [18] - reducing their availability for complex work.

Among various onboarding strategies, code tours—step-by-step guides of critical code sections—have shown promise in improving code navigation and understanding [20]. However, like other documentation artifacts, code tours require manual creation and maintenance by senior developers. Recent advancements in Large Language Models (LLMs) have demonstrated their capability to generate code explanations and assist in onboarding tasks [1, 2, 7, 13, 14, 16, 18], suggesting their potential for automating code tour generation. Our preliminary findings show that LLMs can also produce code tours to explain stack traces [4] to guide debugging, a typical entry task [8].

Nevertheless, most studies rely on state-of-the-art, closed-weight models, raising concerns about reproducibility, stability, and privacy [2, 17]—critical issues for companies handling sensitive codebases. We propose fine-tuning open-weight, locally-runnable LLMs ($\leq$ 10B parameters) to generate high-quality, debugging-focused code tours. Successfully fine-tuning these models requires tackling three key research questions: (*RQ1*) identifying relevant points of interest for debugging, (*RQ2*) adapting an LLM to generate code tours, and (*RQ3*) designing an evaluation framework to assess their quality. Ultimately, we aim to develop, iteratively, a transparent, reproducible, and privacy-preserving approach for automating debugging-oriented code tours, enhancing the onboarding experience for new developers.

2 Background and Related Work

2.1 Onboarding Developers

Documentation and mentoring [10] play a critical role in supporting onboarding. However, in large projects like those at CERN, mentoring junior developers can take up to 30% of senior developers' time [18], reducing their time for other valuable tasks. Debugging is typically assigned to new developers as an onboarding task [8]. Although debugging is a practical learning approach, it assumes that newcomers can efficiently navigate the code. **Code tours** [6] are structured walkthroughs of key code segments that can help navigate and understand a new codebase [20]. However, their creation and maintenance require human effort, which limits scalability. Despite recent studies that have explored automating code tour generation [4], this is an underexplored research area.

2.2 Selecting Relevant Code Segments

Selecting relevant code segments is a key challenge in automating debugging-focused code tours. In prior work [4], we used stack trace analysis for step selection. While effective, this is only one approach among many possible alternatives, leaving other selection strategies largely unexplored. Spectrum-based Fault Localization (SFL) identifies program elements more likely to contain faults and has been successfully applied in debugging scenarios [9]. Hotspot detection [22] prioritizes frequently modified or complex areas of the codebase. Thus, they may be transferable to the context of step selection for debugging-focused code tours.

FSE Companion '25, Trondheim, Norway

2.3 Generating Documentation

Although LLMs improve code comprehension, reduce cognitive load, and generate explanations adapted to user needs, their effectiveness varies depending on user expertise and interaction patterns [1, 13, 16]. In the context of code tours, generated explanations often suffer from verbosity and lack cohesion between steps [4], indicating further work to improve code tour generation. Moreover, a common concern is *confidentiality* [2] when using LLMs to explain proprietary code. Indeed, many studies rely on closed-weight models, which, while powerful, require sending proprietary code to external servers. Besides, concerns about reproducibility and stability have been raised [17]. Open-weight LLMs offer greater control but typically require domain-specific fine-tuning to match the performance of proprietary models.

However, fine-tuning alone does not guarantee alignment with the developer's non-functional needs [15] - such as clarity, efficiency, and scrutability [4]. So, to better align LLM-generated explanations with developer expectations, Reinforcement Learning from Human Feedback (**RLHF**) has been widely used [15], particularly in conversational agents such as ChatGPT. RLHF involves fine-tuning a model by rewarding outputs that align with human preferences on dimensions such as clarity, relevance, and style. While RLHF [15] improves alignment with developer expectations, its reliance on human annotations makes it impractical if you have limited resources. To address RLHF reliance on humans, Reinforcement Learning from AI Feedback (**RLAIF**) leverage **LLM-as-a-Judge** methodologies, which replace human annotations with judgments from larger models, to reduce annotation costs while still improving alignment [3]. For example, CodeUltraFeedback [24] applies the LLM-as-a-Judge [26] methodology to assess alignment with non-functional requirements in code-related tasks, such as readability, complexity, and style—dimensions often challenging to capture using traditional automated metrics. A core component of RLAIF is **knowledge distillation**, where larger LLMs generate synthetic datasets to fine-tune smaller models iteratively [23, 25]. However, careful selection of teacher models is necessary, as many companies impose licensing restrictions that prevent proprietary LLMs from being used to fine-tune competitor models. **Retrieval-Augmented Generation (RAG)** [12] offers an alternative to extensive fine-tuning by dynamically retrieving relevant documents during inference to provide task-specific context. Correia et al. [7] developed a conversational agent that augments LLM responses with project-specific documentation, achieving expert-rated satisfaction levels exceeding human in more than half of the cases.

2.4 Evaluating Quality

Evaluating the quality of generated code tours is a non-trivial challenge, as their usefulness extends beyond traditional documentation metrics. Our prior work [4] has adapted quality dimensions from recommender system research [21] to assess code tours along three key criteria: (1) **Transparency** – Clearly explains how the code works. (2) **Efficiency** – Helps identify and report inaccuracies. (3) **Scrutability** – Facilitates quick navigation and understanding. While these dimensions provide a structured framework, existing evaluations have been limited in scale and consistency. Manual assessments introduce subjectivity, and inter-rater agreement has not been extensively studied in this context. **LLM-as-a-Judge** [26] uses LLMs aligned with human preference as proxies for human annotators. This is already a scalable alternative to human evaluation in related tasks for RLAIF, such as assessing readability and clarity in code [24].

3 Research Approach

This section presents our research approach to automating the generation of debugging-focused code tours. We address three primary research questions (RQs):

RQ1 Given a failing test, how can we identify points of interest relevant to debugging for inclusion in code tours?
RQ2 How can locally-runnable LLMs be adapted to generate contextually relevant explanations of code tour steps?
RQ3 How can we evaluate the quality and usefulness of the generated code tours?

To validate our approach, we will begin with RQ3 (how to evaluate quality), which supports RQ2 (generating better explanations), while RQ1 (step selection) can progress in parallel. We will outline our approach for each research question, discuss prior work, propose future work, detail our evaluation methods, and highlight the expected contributions. If we need participants for an experiment, we will recruit them through the university and alumni channels until we reach saturation.

3.1 RQ1: Selecting Relevant Code Segments

Following our preliminary study [4], where we explored using stack traces to structure code tours, we aim to broaden our selection strategy. However, since our primary focus is to aid debugging for onboarding, our approach will always follow a call path between a failing test and a faulty method—but which path? *That is the question!*

Prior & Future Work. Our previous work [4] leveraged stack traces to structure code tours, as they provide an execution flow from a failing test to a faulty method. While stack traces are valuable debugging aids, it remains unclear whether alternative heuristics can improve coherence and debugging efficiency. In practical debugging scenarios, faults are supposedly unknown. Therefore, we will use *fault localization methods* [9] to identify faulty methods. Then, multiple heuristics will be evaluated to filter paths between the failing test and the potentially faulty method.

Evaluation. We will assess the effectiveness of our path selection strategies using both quantitative and qualitative methods. Quantitatively, we will measure the alignment between selected steps and hotspots [22]. Qualitatively, we will conduct semi-structured interviews with developers and students, asking them to assess the relevance of a sequence of steps in debugging tasks. We remain open to refining our evaluation methods based on early findings or feedback from the research community.

Contribution. Methodologically, our research will improve step selection for structuring code tours that are coherent, practical, and debugging-efficient.

3.2 RQ2: Generating Documentation

Our preliminary study [4], use a close-weight, cloud-based model, `GPT3.5` [5], which raise some concern of stability, reproducibility [17] and, more importantly, privacy [2]. Our approach will be to align locally-runnable LLMs to generate qualitative code tour explanations.

Prior & Future Work. Using `GPT3.5` [5], we generated explanations for the selected steps. Our preliminary analysis revealed two main limitations: (1) explanations often contained redundant or low-level details, and (2) explanations lacked contextual connections across steps. We intend to adapt open-weight, locally-runnable LLMs via RLAIF to enhance explanation quality with minimal human involvement. Additionally, our initial experiments were conducted on `Defects4J` [11], which introduces a risk of the LLM knowing the codebase [17]. This threatens the validity as there could be some overfitting. Thus, we will conduct future experiments using `GitBug-Java` [19] and private codebases to mitigate overfitting risks.

Evaluation. Using the checklist from Section 3.3, we will ask humans and LLMs to assess the generated explanations' quality. We will also involve developers in exploratory sessions combining quantitative and qualitative methods to evaluate how well the generated code tours support understanding and navigation of the codebase. For example, we plan to measure debugging time, completed by surveys on *perceived* clarity and usefulness. We also expect the model to face limitations when detecting, reasoning, and explaining cause-effect chains spanning multiple functions or modules. Thus, our evaluation will include scenarios with more complex interactions. If we observe signs of shortcomings, we should delegate the detection of such interactions to causal reasoning techniques (this would be part of RQ1, then). Thus, the LLM's role would be to explain, in free-form text, the reasoning produced.

Contribution. Methodologically, we will introduce a framework for adapting locally-runnable, open-weight LLMs via Reinforcement Learning from AI Feedback (RLAIF).

3.3 RQ3: Evaluating Quality

We will refine an evaluation checklist to ensure that generated explanations effectively support debugging. Initially, we will collect open-ended evaluations from LLMs based on a root criterion—usefulness for debugging—allowing them to score explanations and justify their ratings. By analyzing these justifications, we should extract recurring evaluation patterns (in a survey coding manner) until saturation is reached, forming an initial set of criteria and a checklist for each.

Prior & Future Work. We developed an initial checklist to assess the quality of generated code tours inspired by evaluation frameworks from recommendation systems [21]. However, this checklist has only been tested with a single rater on a small subset of generated tours, so there is no inter-rater agreement analysis. Our goal is to improve the checklist iteratively. First, we plan to collect LLM-generated assessments to extract recurring evaluation criteria. Once a stable inter-rater agreement is, if possible, reached among LLMs, we will introduce human evaluators to compare their assessments with LLM judgments.

Evaluation. Once a preliminary checklist is established, we will validate it in two stages. First, we measure inter-rater agreement among LLMs to ensure consistency. Once a stable agreement is reached, we will introduce human evaluators and compare their assessments against LLM judgments. This iterative process ensures the checklist captures meaningful quality indicators while minimizing reliance on busy developers.

Contribution. Empirically, we will provide a dataset of LLM-generated code tour explanations, evaluated by LLM and human experts, along with an analysis of the inter-rater agreement to assess the reliability of LLM-based evaluation. Theoretically, we will refine iteratively a checklist for assessing code tour quality, ensuring a structured evaluation framework. Methodologically, we will propose an iterative pipeline for extracting evaluation criteria using LLM-as-a-Judge.

4 Conclusion

This work investigates using Large Language Models (LLMs) to automate debugging-focused code tours to support developer onboarding. We proposed an approach that leverages stack traces for selecting relevant code segments and LLMs for generating step-by-step explanations. Our preliminary results demonstrate that while automatic generation is feasible with `GPT3.5` [5], key limitations persist, such as redundant explanations, low-level details, and a lack of contextual linking between steps. Furthermore, the community has highlighted concerns about closed-weight models, asking for research about the open-weight ones.

To address these challenges, we outlined three main research directions: (1) improving step selection by exploring alternative methods, (2) enhancing explanation generation by aligning locally-runnable LLMs using Reinforcement Learning from AI Feedback (RLAIF), and (3) refining evaluation criteria through an iterative process leveraging both LLM and human assessments.

Our expected contributions include an iterative pipeline to generate and evaluate code tours automatically. To validate these contributions, we plan to conduct empirical studies measuring the effectiveness of different step selection techniques, the impact of fine-tuning strategies on explanation quality, and the reliability of LLM-based evaluation frameworks.

While our preliminary evaluation provides initial insights into the feasibility and limitations of LLM-generated code tours, further work is required to enhance their effectiveness and scalability. We invite feedback from the research community to refine our approach, explore additional validation methodologies, and assess the broader impact of LLM-generated onboarding documentation in real-world settings.

Acknowledgments

This research was supported by the ARIAC project (No. 2010235) funded by the Service Public de Wallonie (SPW Recherche), and the SQUAL.AI project (No. 2025/658876) supported by Wallonie-Bruxelles International (WBI).

References

[1] Elijah Kayode Adejumo and Brittany Johnson. 2024. Towards Leveraging LLMs for Reducing Open Source Onboarding Information Overload. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering,*

ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, New York, NY, USA, 2210–2214. https://doi.org/10.1145/3691620.3695286

[2] Maider Azanza, Juanan Pereira, Arantza Irastorza, and Aritz Galdos. 2024. Can LLMs Facilitate Onboarding Software Developers? An Ongoing Industrial Case Study. In *36th International Conference on Software Engineering Education and Training, CSEE&T 2024, Würzburg, Germany, July 29 - Aug. 1, 2024*. IEEE, New York, NY, USA, 1–6. https://doi.org/10.1109/CSEET62301.2024.10662989

[3] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosiute, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemí Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional AI: Harmlessness from AI Feedback. *CoRR* abs/2212.08073 (2022). https://doi.org/10.48550/ARXIV.2212.08073 arXiv:2212.08073

[4] Martin Balfroid, Benoît Vanderose, and Xavier Devroey. 2024. Towards LLM-Generated Code Tours for Onboarding. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering, NLBSE 2024, Lisbon, Portugal, 20 April 2024*, Maliheh Izadi, Andrea Di Sorbo, and Sebastiano Panichella (Eds.). ACM, New York, NY, USA, 65–68. https://doi.org/10.1145/3643787.3648033

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). Curran Associates, Inc., Red Hook, NY, United States. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[6] Jonathan Carter. 2020. Codetour - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=vsls-contrib.codetour

[7] João Lucas Correia, Morgan C. Nicholson, Daniel Coutinho, Caio Barbosa, Marco Castelluccio, Marco Aurélio Gerosa, Alessandro F. Garcia, and Igor Steinmacher. 2024. Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project. In *Proceedings of the 1st ACM International Conference on AI-Powered Software, AIware 2024, Porto de Galinhas, Brazil, July 15-16, 2024*, Bram Adams, Thomas Zimmermann, Ipek Ozkaya, Dayi Lin, and Jie M. Zhang (Eds.). ACM, New York, NY, USA, 10 – 18. https://doi.org/10.1145/3664646.3664758

[8] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline de Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel (Eds.). ACM, New York, NY, USA, 275–284. https://doi.org/10.1145/1806799.1806842

[9] Higor Amario de Souza, Marcos Lordello Chaim, and Fabio Kon. 2016. Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges. *CoRR* abs/1607.04347 (2016). arXiv:1607.04347 http://arxiv.org/abs/1607.04347

[10] An Ju, Hitesh Sajnani, Scot Kelly, and Kim Herzig. 2021. A Case Study of Onboarding in Software Teams: Tasks and Strategies. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, New York, NY, USA, 613–623. https://doi.org/10.1109/ICSE43902.2021.00063

[11] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, Corina S. Pasareanu and Darko Marinov (Eds.). ACM, New York, NY, USA, 437–440. https://doi.org/10.1145/2610384.2628055

[12] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[13] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 1, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023*, Maureen Doyle, Ben Stephenson, Brian Dorn, Leen-Kiat Soh, and Lina Battestilli (Eds.). ACM, New York, NY, USA, 931–937. https://doi.org/10.1145/3545945.3569785

[14] Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. *CoRR* abs/2307.08177 (2023). https://doi.org/10.48550/ARXIV.2307.08177 arXiv:2307.08177

[15] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). Curran Associates, Inc., Red Hook, NY, United States. https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

[16] Jonan Richards and Mairieli Wessel. 2024. What You Need is what You Get: Theory of Mind for an LLM-Based Code Understanding Assistant. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2024, Flagstaff, AZ, USA, October 6-11, 2024*. IEEE, New York, NY, USA, 666–671. https://doi.org/10.1109/ICSME58944.2024.00070

[17] June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the Silence: the Threats of Using LLMs in Software Engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, NIER@ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, New York, NY, USA, 102–106. https://doi.org/10.1145/3639476.3639764

[18] Ioan Cristian Schuszter and Marius Cioca. 2024. Increasing the Reliability of Software Systems Using a Large-Language-Model-Based Solution for Onboarding. *Inventions* 9, 4 (2024), 79. https://doi.org/10.3390/inventions9040079

[19] André Silva, Nuno Saavedra, and Martin Monperrus. 2024. GitBug-Java: A Reproducible Benchmark of Recent Java Bugs. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, Diomidis Spinellis, Alberto Bacchelli, and Eleni Constantinou (Eds.). ACM, New York, NY, USA, 118–122. https://doi.org/10.1145/3643991.3644884

[20] Grace Taylor and Steven Clarke. 2022. A Tour Through Code: Helping Developers Become Familiar with Unfamiliar Code. In *Proceedings of the 33rd Annual Workshop of the Psychology of Programming Interest Group, PPIG 2022, The Open University, Milton Keynes, UK & Online, September 5-9, 2022*, Simon Holland, Marian Petre, Luke Church, and Mariana Marasoiu (Eds.). Psychology of Programming Interest Group, New York, NY, USA, 114–126. https://ppig.org/papers/2022-ppig-33rd-taylor/

[21] Nava Tintarev and Judith Masthoff. 2015. Explaining Recommendations: Design and Evaluation. In *Recommender Systems Handbook*, Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). Springer, New York; London, 353–382. https://doi.org/10.1007/978-1-4899-7637-6_10

[22] Adam Tornhill. 2024. *Your Code as a Crime Scene : Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs*. Pragmatic Bookshelf, Flower Mound, TX, USA. https://www.torrossa.com/en/resources/an/5241725

[23] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Stroudsburg, PA, USA, 13484–13508. https://doi.org/10.18653/V1/2023.ACL-LONG.754

[24] Martin Weyssow, Aton Kamanda, and Houari A. Sahraoui. 2024. CodeUltraFeedback: An LLM-as-a-Judge Dataset for Aligning Large Language Models to Coding Preferences. *CoRR* abs/2403.09032 (2024). https://doi.org/10.48550/ARXIV.2403.09032 arXiv:2403.09032

[25] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering Large Pre-Trained Language Models to Follow Complex Instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=CfXh93NDgH

[26] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). Curran Associates, Inc., Red Hook, NY, United States. http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html