Program Merge: What's Deep Learning Got to Do with It?

A DISCUSSION WITH SHUVENDU LAHIRI, ALEXEY SVYATKOVSKIY, CHRISTIAN BIRD, ERIK MEIJER AND TERRY COATTA

If you regularly work with open-source code or produce software for a large organization, you're already familiar with many of the challenges posed by collaborative programming at scale. Some of the most vexing of these tend to surface as a consequence of the many independent alterations inevitably made to code, which, unsurprisingly, can lead to updates that don't synchronize.

Difficult merges are nothing new, of course, but the scale of the problem has gotten much worse. This is what led a group of researchers at MSR (Microsoft Research) to take on the task of complicated merges as a grand program-repair challenge—one they believed might be addressed at least in part by machine learning.

To understand the thinking that led to this effort and then follow where that led, we asked Erik Meijer and Terry Coatta to speak with three of the leading figures in the MSR research effort, called DeepMerge ("DeepMerge: Learning to Merge Programs," Microsoft Research). *Meijer was long a member of MSR, but at the time of this discussion was director of engineering at Meta. Coatta is the CTO of Marine Learning Systems. Shuvendu Lahiri and Christian Bird, two of the researchers who helped drive this effort, represent MSR, as does Alexey Svyatkovskiy, who was with Microsoft DevDiv (Development Division) at the time.*

TERRY COATTA What inspired you to focus on merge conflicts in the first place? And what made you think you'd be able to gain some advantage by applying AI techniques?
CHRISTIAN BIRD Back in the winter of 2020, some of us started talking about ways in which we might be able to use machine learning to improve the state of software engineering. We certainly thought the time was right to jump into an effort along these lines in hopes of gaining enough competency to launch into a related research program.

We tried to identify problems other researchers weren't already addressing, meaning that something like code completion—which people had been working on for quite some time—was soon dismissed. Instead, we turned to problems where developers didn't already have much help.

Shuvendu [Lahiri] has a long history of looking at program merge from a symbolic perspective, whereas my own focus has had more to do with understanding the changes that occur in the course of program merges. As we were talking about this, it dawned on us that almost no one seemed to be working on program merge. And yet, that's a problem where we, as developers, still have little to rely upon. For the most part, we just look at the diffs between different generations of code to see if we can figure out exactly what's going on. But there just isn't much current tooling to help beyond that, which can prove to be problematic whenever there's a merge conflict to resolve.

So, we figured, "OK, let's look at how some deep-learning models might be applied to this problem. As we go along, we'll probably also identify some other things we can do to build on that."

SHUVENDU LAHIRI Yes, as Chris suggests, I've been thinking about the issues here for quite some time. Moreover, we found program merge to be appealing, since it's a collaboration problem. That is, even if two skilled developers make correct changes, the merge itself may introduce a bug.

We were also keenly aware of the sort of pain program-merge problems can cause, having known about it through studies within Microsoft *("What developers want and need from program analysis: an empirical study,"* Microsoft Research*).* I thought maybe there was something we could do to provide relief. It also turns out that AI was just coming along at that point, and Alexey [Svyatkovskiy] had already developed a couple of powerful models that looked quite promising for code completion. What's more, information about merge conflicts was just starting to become more readily available from the Git commit history, so that too looked like it might serve as a good upfront source of clean data.

ERIK MEIJER I like the fact that you focused on merge conflict, since, when it comes to this, I don't think source control solves any of the real problems. Maybe I'm being a little extreme here, but even if source control lets you know where you have a merge conflict, it won't help you when it comes to actually resolving the conflict. In fact, I'm baffled as to why this problem wasn't solved in an intelligent manner a long time ago. Are people just not listening to complaints from actual users?

SL Basically, I think it comes down to academicians consistently resorting to symbolic methods to solve this problem. Whereas people in the real world have

looked at this as just another aspect of programming, practitioners have been more inclined to approach it as a social process—that is, as a problem best addressed by encouraging co-workers to figure out solutions together. Personally, I've always seen merge conflicts as more of a tooling challenge.

ALEXEY SVYATKOVSKIY For me, this just looked like an exciting software engineering problem to address with machine learning. I've spent years working on code completion, but this effort looked like something that would take that up to the next level of complexity, since it necessarily would involve aligning multiple sequences somehow and then complementing that with an understanding of where things ought to be inserted, deleted, or swapped. And, of course, there were also those special cases where the developer would be able to add new tokens during the merge.

This took us on a journey where we ended up addressing program merge down at the line-and-token level. I found this fascinating, since a lot of people don't have any idea about how merge actually works and so, by extension, don't have a clear understanding about what leads to merge conflicts. Taking on this problem also seemed important in that, while merge conflicts are far less common than software bugs, they require considerably more time to resolve and can end up causing far more pain.

TC How did you initially attack the problem?

SL We realized that repositories (both open-source ones on GitHub and internal ones at Microsoft) contain data on merge conflicts and their resolution across several different programming languages. What's more, Alexey

had recently created a neural model that had been pre-trained on a large subset of the code in those repositories. Our initial thought was to fine-tune that model with information about merge conflicts and their resolution. And we figured that should be simple enough to be treated as an intern project. So, that's how we scoped it initially. We figured: Just get the data, train a model, and deploy. What we failed to grasp was that, while there was an ample amount of program merge data to be mined, coming to an understanding of what the intent behind all those merges had been was at least as important as the data itself. In fact, it proved to be absolutely critical.

We concluded that if you manage to combine your existing merge tools correctly, add just the right amount of granularity and then employ neural modeling, you can often succeed in reducing the complexity.

—Shuvendu Lahiri

A considerable amount of time and effort was required to understand and interpret the data. This included some significant technical challenges. For example, how best to align these programs? And how can you communicate to a neural model that these are not independent programs but instead represent some number of changes to an underlying program? The notion of how to go from program text to a program edit became quite crucial and, in fact, required considerable research. Ultimately, we concluded that if you manage to combine your existing merge tools correctly, add just the right amount of granularity—which for us proved to be tokens—and then employ neural modeling, you can often succeed in reducing the complexity. But it took us quite a bit of time to work that out.

Of course, we also underestimated the importance of user experience. How exactly would a user end up employing such a tool—one that's AI-based, that is? And what would be the right time to surface that aspect of the tool?

TC I find it fascinating that it proved to be so difficult to scope this project correctly. Can you dig a bit deeper into that?

AS To me, at least, as we were analyzing the different types of merges, it soon became clear that there are varying levels of complexity. Sometimes we'd find ourselves looking at two simple merge resolution strategies where it essentially came down to "Take ours or take theirs." Such cases are trivial to analyze, of course, and developers don't require much AI assistance when it comes to resolving these conflicts.

But then there's another class of merge where a new interleaving line is introduced that involves more than just concatenation. There could also be token-level interleaving, where lines in the code have been broken and new tokens introduced in between. This leads to the notoriously complex case where a switch to token-level granularity proves to be crucial. Beyond that, there's a whole other class of merges where you find somebody has introduced some new tokens.

EM How do you go about defining what you consider to be a correct merge? Doesn't that require you to make your own value judgments in some sense?

SL Well, I'll just say we had a very semantic way of looking at merges. Essentially: "Forget about the syntax; instead, what does it *mean* for the merge to be correct?" In effect, this amounts to: If something was changed in one program, then that ought to be reflected in the merge. And, if that also changes a behavior, then that too ought to be included in the merge. But no other changes or altered behaviors should be introduced.

We then found, however, that we could get tangled up whenever we ran into one of these "take my changes or take yours" merges. We also found that one set of changes would often just be dropped—like a branch being deprecated, as Alexey once pointed out. This is how we discovered that our initial notion of correctness didn't always hold. That's also how we came to realize we shouldn't adhere to overly semantic notions.

So, we decided just to do our best to curate the training data by removing any indications of "take your changes or take mine" wherever possible. Then we looked at those places where both changes had been incorporated to some extent and said, "OK, so this now is our ground truth—our notion of what's correct." But notice that this is *empirical* correctness as opposed to *semantic* correctness—which is to say, we had to scale back from our original high ambitions for semantic correctness.

AS We now treat user resolutions retrieved from the GitHub commit histories as our ground truth. But yes, naturally, there are all kinds of ways to define a "correct" merge. For example, it's possible to reorder the statements in a structured merge and yet still end up with a functionally equivalent resolution. Yet that would be deemed as "incorrect," so, there's clearly room for retooling our definition of correctness. In this instance, however, we chose to take a data-driven approach that treats user resolutions from the GitHub commit histories as our ground truth.

CB Right. And let me also say that, from the beginning of this project, we decided to approach it as something

that might yield a product. With that in mind, we realized it needed to be, perhaps not language-agnostic, but at least something that could be readily adapted to multiple languages—and definitely *not* something that would require some bespoke analysis framework for each language. That essentially guided our choice not to employ richer or more complex code representations.

EM I've also run into situations like this where it looked really tempting to use an AST [abstract syntax tree] or something of the sort, since that would provide all the structure that was required. But then, as you go deeper into that sort of project, you find yourself wondering whether it's actually a good idea to feed semantically rich programs into models and start thinking it might be better just to send strings instead.

TC To dive a bit deeper into that, you had a practical motivation to work with a token-based approach. But what does your intuition tell you about how models behave when you do that? If you feed a model a rich, structured information set, is that model then actually more likely to make better decisions? Or is that perhaps a false assumption?

If you feed a model a rich structured information set, is that model then actually more likely to make better decisions? Or is that perhaps a false assumption?

—Terry Coatta

AS The model ought to be able to make better decisions, I think.

EM All right, but can I challenge that a bit? The model can handle the syntax and semantic analysis internally, which suggests this work may not need to be done ahead of time, since machines don't look at code the same way humans do. I don't know why the model couldn't just build its own internal representation and then let a type-checker come along at the end of the process.

CB I think it's hazardous to speculate about what models may or may not be capable of. I mean, that's a super-nuanced question in that it depends on how the model has been trained and what the architecture is like—along with any number of other things. I'm constantly surprised to learn what models are now capable of doing. And, in this case, we're talking about the state of the world back in 2020—even as I now find it hard to remember what the state of the world looked like six months prior to the GPT models becoming widespread.

SL For one thing, we were using pretrained models to handle classification and generation, which then left us with quite a bit of work to do in terms of representing the resulting edits at the AST level before tuning for performance. That certainly proved to be a complex problem—and one that came along with some added computational costs. Also, as I remember it, the models we were using at the time had been trained as text representations of code—meaning we then needed to train them on a lot more AST-level representations to achieve better performance. I'm sure it would be fascinating to go back to revisit some of the decisions we made back in 2020.

EM What model are you using now?

AS For this iteration, we're employing a token-level merge along with a transformer-based classifier. We've also been looking at using a prompt-driven approach based on GPT-4.

EM I love that this is now something where you can take advantage of demonstrated preferences to resolve merge conflicts instead of being left to rely solely on your own opinions.

SL Another way of looking at this that came up during

our user studies was that, even after a merge has been produced, someone might want to know why the merge was accomplished in that particular way and may even want to see some evidence of what the reasoning was there. Well, that's a thorny issue.

But one of the nice things about these large foundational models is that they're able to produce textual descriptions of what they've done. Still, we haven't explored this capability in depth yet, since we don't actually have the means available to us now to evaluate the veracity of these descriptions. That will have to wait until some user studies supply us with more data. Still, I think there are some fascinating possibilities here that ultimately should enable us to reduce some of the friction that seems to surface whenever these sorts of AI power tools are used to accomplish certain critical tasks.

In the event you regularly work with open-source code, you're surely already familiar with some of the challenges that can arise in the course of trying to resolve merge conflicts. Many of these problems have been encountered for as long as people have collaborated on programs, and these have metastasized as the scale and complexity of software has multiplied many times over. Also, with thousands of developers sometimes now collaborating on projects, the potential for conflicts only continues to soar.

Many of these are conflicts that can lead to program failures, of course. But even worse in some respects are the more subtle semantic merge conflicts that can fail the compiler, break a test, or introduce a regression. Despite

these painfully obvious problems, the program merge issue has been largely left to fester for decades simply because the challenge of addressing it has seemed so daunting.

TC You've mentioned that you had access to a vast amount of training data, but you've also suggested some of that data contained surprises—which is to say it proved to be both a blessing and a curse. Can you go into that a bit more?

SL Yes, we were surprised to find that a large percentage of the merges—perhaps 70 percent—had the attribute of choosing just one side of the edit and then dropping the other. In some of those cases, it seemed one edit was superseding the others, but it can be hard to be sure whenever the syntax changes a little. In many instances, there were genuine edits that had been dropped on the floor. It was unclear whether that was due to a tooling problem or a social issue—that is, in some cases, perhaps some senior developer's changes had superseded those that had been made by a junior developer. Another hypothesis was that, instead of a single merge, some people may have chosen to merge in multiple commits.

This sort of thing was so common that it accounted for a significant portion of the data, leaving us uncertain at first as to whether we should throw out these instances, ignore them, or somehow make an effort to account for them. That certainly proved to be one of the bigger surprises we encountered.

Another surprise was that we discovered instances where some new tokens had been introduced that were irrelevant to the merge. It was unclear at first whether

We learned that some users still wanted to be able to use the approach that had been dismissed. We solved that problem by providing a "B option" that people could get to by using a drop-down menu.

—Alexey Svyatkovskiy

those were due to a genuine conflict in the merge or just because somebody had decided to add a pretty print statement while doing the refactoring. That proved to be another thorny issue for us.

TC How did you resolve that? It sounds like you had some datasets you didn't quite know how to interpret. So, how did you decide what should be classified as correct merges or treated as incorrect ones?

SL We curated a dataset that did not include the "trivial" merge resolutions, with the goal of assisting users with the more complex cases first. As Alexey mentioned, users may not need tooling support for those resolutions that only require dropping one of the two edits.

AS And then, from user studies, we learned that some users still wanted to be able to use the approach that had been dismissed. We solved that problem by providing a "B option" that people could get to by using a drop-down menu.

SL Which is to say we addressed the problem by way of user experience rather than by changing the model.

The other data problem we encountered had to do with new tokens that would occasionally appear. Upon closer examination, we found these tokens were typically related to existing changes. By going down to token-level merges, we were able to make many of these aspects go away. Ultimately, we built a model that excluded that part of the dataset where new tokens were introduced.

EM In terms of how you went about your work, I understand one of the tools you particularly relied on was Tree-sitter [a parser-generator tool used to build syntax trees]. Can you tell us a bit about the role it played in your overall development process?

CB We were immediately attracted to Tree-sitter because it lets you parse just about anything you can imagine right off the shelf. And it provides a consistent API, unlike most other parsers out there that each come with their own API and work only with one language or another.

For all that, I was surprised to learn that Tree-sitter doesn't provide a tokenizing API. As an example of why that proved to be an issue for us, we wanted to try Python, which basically lets everyone handle their own tokenizing. But, of course, Tree-sitter didn't help there. We resorted to a Python tokenizing library.

Beyond that relatively small complaint, Tree-sitter is great in terms of letting you apply an algorithm to one language and then quickly scale that up for many other languages. In fact, between that capability and the Python tokenizing library, which made it possible for us to handle multiple languages, we were able to try out things with other languages without needing to invest a lot of upfront effort. Of course, there's still the matter of obtaining all the data required to train the model, and that's always a challenge. At least we didn't need to write our own parsers, and the consistent interfaces have proved to be incredibly beneficial.

EM Once you finally managed to get all this deployed, what turned out to be your biggest surprise?

CB There were so many surprises. One I particularly remember came up when we were trying to figure out how people would even want to view merge conflicts and diffs. At first, some of us thought they'd want to focus only on the conflict itself—that is, with a view that let them see both their side and the other side. It turns out you also need to be able to see the base to understand the different

implications between an existing branch in the base and your branch.

So, we ran a Twitter survey to get a sense of how much of that people thought we should show. How much of that did they even *want* to see? For example, as I recall, most people couldn't even handle the idea of a three-way diff, or at least weren't expecting to see anything quite like that. That really blew my mind, since I don't know how anyone could possibly expect to deterministically resolve a conflict if they don't know exactly what they're facing.

Some other issues also came up that UI people probably would expect, but I nevertheless was incredibly surprised. That proved to be a big challenge, since we'd been thinking throughout this whole process that we'd just get around to the UI whenever we got around to it. And yes, as this suggests, our tendency initially was just to focus on making sure the underlying algorithm worked. But then we found to our surprise just how tough it could be to find the right UI to associate with that.

TC From what you say, it seems you weren't surprised about the need for a good user experience, but it did surprise you to learn what's considered to *be* a good experience. What are your thoughts now on what constitutes a good user experience for merge?

CB I'm not entirely clear on that even now, but I'll be happy to share some of the things we learned about this early on. As we've already discussed, people definitely want to see both sides of a merge. Beyond that, we discovered that they want the ability to study the provenance of each part of the merge because they want to know where each token came from.

So, we wrote some code to track each token all the

way back to whichever side it came from.

There also were tokens that had come in from both sides. To make it clear where a token had originated, we wrestled with whether we should add colors as an indicator of that. How might that also be used to indicate whether a token happens to come from both sides or simply is new?

In addition, we knew it was important that the interface didn't just ask you to click "yes" or "no" in response to a suggested change, since it's rare to find any merge that's going to be 100 percent correct. Which is to say developers are going to want to be able to modify the code and will only end up being frustrated by any interface that denies them that opportunity.

We know that if we offer three suggestions for a merge rather than just one, the chance of the best one being selected is much higher.

—Christian Bird

The real challenge is that there are lots of moving pieces in any given merge. Accordingly, there are many possible views, and yet you still want to keep things simple enough to avoid overwhelming the user. That's a real challenge. For example, we know that if we offer three suggestions for a merge rather than just one, the chance of the best one being selected is much higher. But that also adds complexity, so we ultimately decided to go with suggesting the most likely option, even though that might sometimes lead to less-optimal results.

There are some other user-experience considerations worth noting. For example, if you are working on some particular Visual Studio feature, you're going to want to produce something that feels intuitive to someone who has been using that same tool. Suffice it to say, there's plenty to think about in this respect. Basically, once you finally get your model to work, you might not even be halfway home,

since that's just how critical—and time-consuming—the user-experience aspect of this work can be.

Yes, user experience actually does matter—even when the users happen to be developers. Accordingly, a substantial user study was launched in this instance, where the subjects of the study were members of MSR's own technical staff.

Another interesting aspect was that the study participants were presented with code samples extracted from their own work. The significance of this, of course, was that it involved the use of not only real-world examples but also ones where all the tradeoffs and implications associated with each important decision point were sure to be fully appreciated by the study subjects.

Which is to say that the exercise proved to be an interesting learning experience for all parties involved.

TC We all know that creating a tool for internal purposes is one thing, while turning that into a product is something else altogether. It seems you took that journey here, so what were some of the bigger surprises you encountered along the way?

SL Actually, we don't have a product yet that implements the DeepMerge algorithm and aren't at liberty to talk about how that might be used in future products. Still, as we've just discussed, I can say most of the unusual challenges we encountered were related to various aspects of the user experience. So, we got much deeper into that here than we normally would.

One of the biggest challenges had to do with determining

how much information needed to be surfaced to convince the user that what was just done was even possible—never mind appropriate. Suddenly you've just introduced some new tokens over here, along with a new parsable tree over there. I think that can really throw some users off.

CB What did all this look like from the DevDiv perspective, Alexey? You deal with customers all the time. What proved to be the biggest challenges there?

AS Some of the most crucial design decisions came down to choosing between client-side or server-side implementation. Our chief concern had to do with the new merge algorithm we were talking about earlier. Customer feedback obtained from user studies and early adopters proved to be particularly crucial in terms of finding ways to smooth things out. Certainly, that helped in terms of identifying areas where improvements were called for, such as achieving better symmetries between what happens when you merge A to B versus when you merge B to A.

SL I'd like to add a couple of points. One is that some developers would prefer to handle these merges themselves. They just don't see the value of tooling when it's used to deal with something they could do themselves. But that just resulted in some inertia, which is always hard to overcome without a lot of usage. Still, from our empirical study we learned that, even when merges were not identical to the ground truth, users would accept them if they proved to be semantically equivalent. Ultimately, that proved to be a pleasant surprise, since it revealed we had previously been undercounting our wins according to our success metrics.

TC Did anything else interesting surface along the way?

CB At one point, one of our interns did a user study that pulled merge conflicts and their resolutions out of Microsoft's historical repositories so they could then be compared with the resolutions our tool would have applied. As you might imagine, quite a few differences surfaced. To understand where our tool may have gone wrong, we went back to consult with those people who had been involved in the original merges and showed them a comparison between what they had done and how the tool had addressed the same merge conflicts.

We specifically focused on those conflicts that had been resolved over the preceding three months on the premise that people might still recall the reasoning behind those decisions. We learned a ton by going through that particular exercise. One of the lessons was that we had probably undercounted how often we were getting things right, since some of these developers would say things like, "Well, this may not exactly match the merge I did, but I would have accepted it anyway."

The other major benefit of that study was the insight it provided into what the user experience for our tool should be. This all proved to be a major revelation for me, since it was the first time I'd been involved in a user study that was approached in quite this way—where developers were pulled in and presented with code they'd actually worked on.

Which is just to say this wasn't at all like one of those lab studies where people are presented with a toy problem. In this case, we were pulling in real-world merge conflicts and then talking with the developers who had worked to resolve them. We learned so much from taking this approach that I'd recommend other researchers consider

doing their own studies in much the same way.

AS Another thing that came out of these user studies was the importance of explainability. With large language models, for example, we can drill into specific three-way diffs and their proposed resolutions and then ask for summaries of certain decisions, which can be helpful when it comes to building confidence in some of these AI suggestions.

Also, as Chris indicated, even when users chose not to go with the solution offered by DeepMerge, the reasoning behind the suggestion still seemed to inform their own thinking and often led to an improved merge resolution.

TC What's next?

SL There's room for more prompt engineering in terms of determining what goes into the model's input. We also need to address correlated conflicts. So far, we've addressed each conflict as if it was independent, but you can have multiple conflicts in a file that all relate to a certain dependency. Some users have told us that, once a resolution has been worked out for one of those conflicts, they'd like to see something similar applied to each of the other conflicts that exhibit a similar pattern, which certainly seems quite reasonable.

Also, while the types of conflicts we've addressed so far are highly syntactic in nature, there is, in fact, a whole spectrum of merge conflicts. There's still much to address, including silent merges that include semantic conflicts, which are much harder to deal with than anything we've handled so far. Still, I'd say it feels like we're off to a reasonably good start.