

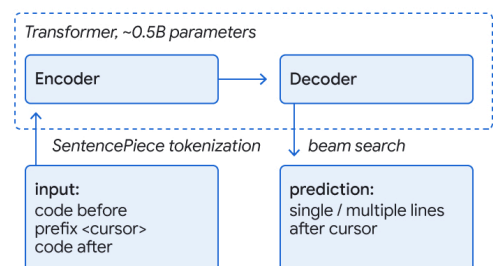
Research

[Home](#) [Blog](#)

ML-Enhanced Code Completion Improves Developer Productivity

July 26, 2022

Posted by Maxim Tabachnyk, Staff Software Engineer and Stoyan Nikolov, Senior Engineering Manager, Google Research



Research

Share

Update — 2022/09/06: This post has been updated to remove a statement about an observed reduction in context switches that could not be confirmed with statistical significance.

The increasing complexity of code poses a key challenge to productivity in software engineering. [Code completion](#) has been an essential tool that has helped mitigate this complexity in [integrated development environments](#) (IDEs). Conventionally, code completion suggestions are implemented with rule-based [semantic engines](#) (SEs), which typically have access to the full repository and understand its semantic structure. Recent research has demonstrated that large language models (e.g., [Codex](#) and [PaLM](#)) enable longer and more complex code suggestions, and as a result, useful products have emerged (e.g., [Copilot](#)). However, the question of how code completion powered by machine learning (ML) impacts developer productivity, beyond [perceived productivity](#) and accepted suggestions, remains open.

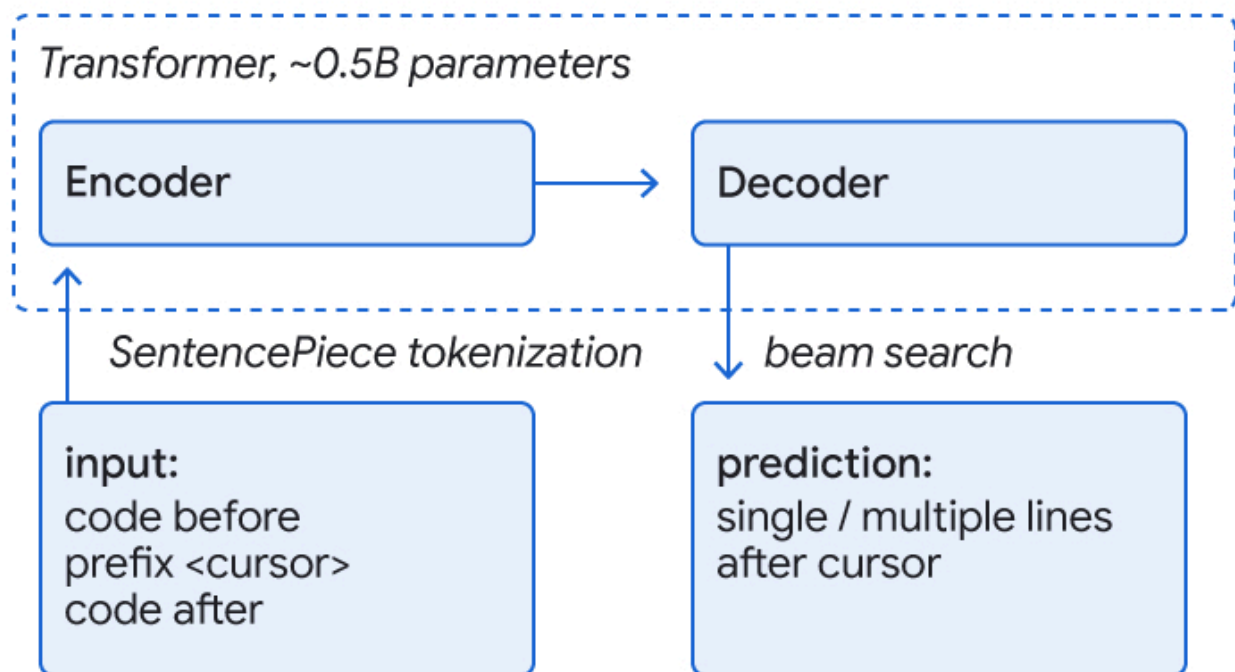
Today we describe how we combined ML and SE to develop a novel [Transformer](#)-based hybrid semantic ML code completion, now available to internal Google developers. We discuss how ML and SEs can be combined by (1) re-ranking SE single token suggestions using ML, (2) applying single and multi-line completions using ML and checking for correctness with the SE, or (3) using single and multi-line continuation by ML of single token semantic suggestions. We compare the hybrid semantic ML code completion of 10k+ Googlers (over three months across eight programming languages) to a control group and see a 6% reduction in coding iteration time (time between builds and tests) when exposed to single-line ML completion. These results demonstrate that the combination of ML and SEs can improve developer productivity. Currently, 3% of new code (measured in characters) is now generated from accepting ML completion suggestions.

Transformers for Completion

A common approach to code completion is to train transformer models, which use a [self-attention](#) mechanism for language understanding, to enable code understanding and completion predictions. We treat code similar to language, represented with sub-word tokens and a [SentencePiece](#) vocabulary, and use encoder-decoder transformer models running on [TPUs](#) to make completion predictions. The input is the code that is

Research

During training on Google's [monorepo](#), we mask out the remainder of a line and some follow-up lines, to mimic code that is being actively developed. We train a single model on eight languages (C++, Java, Python, Go, Typescript, Proto, Kotlin, and Dart) and observe improved or equal performance across all languages, removing the need for dedicated models. Moreover, we find that a model size of ~0.5B parameters gives a good tradeoff for high prediction accuracy with low latency and resource cost. The model strongly benefits from the quality of the monorepo, which is enforced by guidelines and reviews. For multi-line suggestions, we iteratively apply the single-line model with learned thresholds for deciding whether to start predicting completions for the following line.



Encoder-decoder transformer models are used to predict the remainder of the line or lines of code.

Re-rank Single Token Suggestions with ML

While a user is typing in the IDE, code completions are interactively requested from the ML model and the SE simultaneously in the backend. The SE typically only predicts a single token. The ML models we use predict multiple tokens until the end of the line, but we only consider the first token to match predictions from the SE. We identify the top

Research

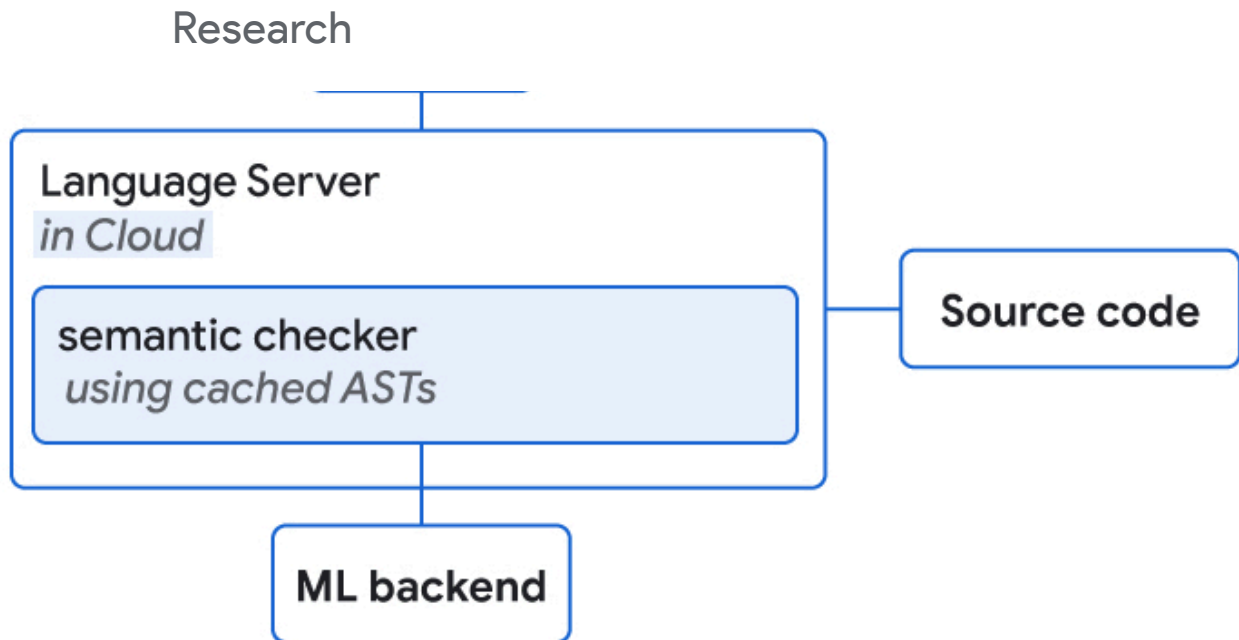
In practice, our SEs are running in the cloud, providing language services (e.g., semantic completion, diagnostics, etc.) with which developers are familiar, and so we collocated the SEs to run on the same locations as the TPUs performing ML inference. The SEs are based on an internal library that offers compiler-like features with low latencies. Due to the design setup, where requests are done in parallel and ML is typically faster to serve (~40 ms median), we do not add any latency to completions. We observe a significant quality improvement in real usage. For 28% of accepted completions, the rank of the completion is higher due to boosting, and in 0.4% of cases it is worse. Additionally, we find that users type >10% fewer characters before accepting a completion suggestion.

Check Single / Multi-line ML Completions for Semantic Correctness

At inference time, ML models are typically unaware of code outside of their input window, and code seen during training might miss recent additions needed for completions in actively changing repositories. This leads to a common drawback of ML-powered code completion whereby the model may suggest code that looks correct, but doesn't compile. Based on internal user experience research, this issue can lead to the erosion of user trust over time while reducing productivity gains.

We use SEs to perform fast semantic correctness checks within a given latency budget (<100ms for end-to-end completion) and use cached [abstract syntax trees](#) to enable a "full" structural understanding. Typical semantic checks include reference resolution (i.e., does this object exist), method invocation checks (e.g., confirming the method was called with a correct number of parameters), and assignability checks (to confirm the type is as expected).

For example, for the coding language [Go](#), ~8% of suggestions contain compilation errors before semantic checks. However, the application of semantic checks filtered out 80% of uncompileable suggestions. The acceptance rate for single-line completions improved by 1.9x over the first six weeks of incorporating the feature, presumably due to increased user trust. As a comparison, for languages where we did not add semantic checking, we only saw a 1.3x increase in acceptance.



Language servers with access to source code and the ML backend are collocated on the cloud. They both perform semantic checking of ML completion suggestions.

Results

With 10k+ Google-internal developers using the completion setup in their IDE, we measured a user acceptance rate of 25-34%. We determined that the transformer-based hybrid semantic ML code completion completes >3% of code, while reducing the coding iteration time for Googlers by 6% (at a 90% confidence level). The size of the shift corresponds to typical effects observed for transformational features (e.g., key framework) that typically affect only a subpopulation, whereas ML has the potential to generalize for most major languages and engineers.

Fraction of all code added by ML	2.6%
Reduction in coding iteration duration	6%
Acceptance rate (for suggestions visible for >750ms)	25%
Average characters per accept	21

Key metrics for single-line code completion measured in production for 10k+ Google-internal developers using it in their daily development across eight languages.

Research

Key metrics for multi-line code completion measured in production for 5k+ Google-internal developers using it in their daily development across eight languages.

Providing Long Completions While Exploring APIs

We also tightly integrated the semantic completion with full line completion. When the dropdown with semantic single token completions appears, we display inline the single-line completions returned from the ML model. The latter represent a continuation of the item that is the focus of the dropdown. For example, if a user looks at possible methods of an API, the inline full line completions show the full method invocation also containing all parameters of the invocation.

```

292
293 absl::Time TimeFromTimespec(timespec ts) {
294     return time_internal::
295 }
296
297 absl::Time TimeFromTimeval(timeval tv) {

```

Integrated full line completions by ML continuing the semantic dropdown completion that is in focus.

```

}
constexpr Duration FromInt64(int64_t v, std::ratio<3600>) {
    return (v <= (std::numeric_limits<int64_t>::max)() / 3600 &&
            v >= (std::numeric_limits<int64_t>::min)() / 60)
        ? MakeDuration(v * 3600)
        : v > 0 ? InfiniteDuration() : -InfiniteDuration();
}

```

Suggestions of multiple line completions by ML.

Conclusion and Future Work

We demonstrate how the combination of rule-based semantic engines and large language models can be used to significantly improve developer productivity with better code completion. As a next step, we want to utilize SEs further, by providing extra information to ML models at inference time. One example can be for long predictions to go back and forth between the ML and the SE, where the SE iteratively checks

Research

Acknowledgements

This research is the outcome of a two-year collaboration between Google Core and Google Research, Brain Team. Special thanks to Marc Rasi, Yurun Shen, Vlad Pchelin, Charles Sutton, Varun Godbole, Jacob Austin, Danny Tarlow, Benjamin Lee, Satish Chandra, Ksenia Korovina, Stanislav Pyatykh, Cristopher Claeys, Petros Maniatis, Evgeny Gryaznov, Pavel Sychev, Chris Gorgolewski, Kristof Molnar, Alberto Elizondo, Ambar Murillo, Dominik Schulz, David Tattersall, Rishabh Singh, Manzil Zaheer, Ted Ying, Juanjo Carin, Alexander Froemmgen, Maxim Kachurovskiy, and Marcus Revaj for their contributions.

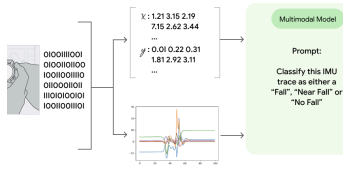
Labels:

Human-Computer Interaction and Visualization

Machine Intelligence

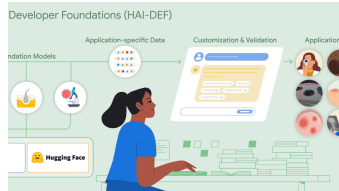
Natural Language Processing

Other posts of interest



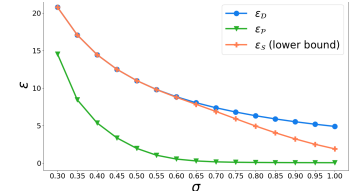
NOVEMBER 25, 2024

Unlocking the power of time-series data with



NOVEMBER 25, 2024

Helping everyone build AI for healthcare applications with



NOVEMBER 25, 2024

Bridging the gap in differentially private model training



*Generative AI ·
Machine Intelligence ·
Machine Perception*

*Health & Bioscience ·
Machine Intelligence ·
Open Source Models &
Datasets*

*Parallel Computing ·
Machine Intelligence ·
Security, Privacy and
Abuse Prevention*

Follow us



Google

About Google

Google Products

Privacy

Terms



Help

Submit feedback