# SPEAK YOUR MIND: INTRODUCING APTLY, THE SOFTWARE PLATFORM THAT TURNS IDEAS INTO WORKING APPS

**David Y.J. Kim[1], Ashley Granquist[1], Evan Patton[1], Mark Friedman[2], Hal Abelson[1]**

*[1]Massachusetts Institute of Technology (USA)*
*[2]Independent Researcher (USA)*

## Abstract

MIT Aptly is a tool that uses the technology of large language models to automatically generate mobile apps from written or spoken natural language descriptions. Similar to Github's Copilot, it is based on OpenAI's Codex, a specially tuned version of GPT-3. Aptly lets people create programs without requiring any use of coding or knowledge of programming. For example, one can tell Aptly by speaking or typing:

> *Make an app with a text box, a list of six languages and a button that says "translate." When the button is clicked, translate the text into the selected language and show the translation.*

The result is a complete functioning app for Android or iPhone. The app has a field for user input and six buttons labeled English, Spanish, French, German, Italian, Japanese. Pressing one of the buttons translates the input to the corresponding language. Aptly's app generation is more than just a syntactic transformation of the input text. Aptly draws upon a large body of code with which it has been trained to provide a context for its app creation. In the above example, Aptly has independently chosen the six languages, something that was not specified in the input text.

As most large language models do, Aptly's performance depends on the input given to OpenAI's Codex. These inputs are referred to as *prompts*. Aptly crafts a prompt by providing a set of example pairs (a textual description of an example app and its corresponding code) along with the description of the desired app. Such prompt engineering is referred to as few-shot prompts. In order to optimize Aptly's performance, when selecting example pairs, we choose the ones that are semantically close to the description of the desired app.

The prospect of no-code platforms is currently sparking considerable ferment in enterprises concerned with professional programming careers. Similarly, Aptly poses challenges for research in computational thinking education for K-12 students. Much of the present-day curriculum emphasizes implementing computational artifacts using text-based coding with Python or block-based coding with Scratch or App Inventor. What will be the foundations for that curriculum when tools like Aptly are common and the transition from ideas to running programs can be accomplished automatically? This presentation will demonstrate Aptly's preliminary performance and review its implementation, which incorporates OpenAI Codex, Amazon Alexa and MIT App Inventor.

Keywords: computational thinking, mobile application, large language model, innovation.

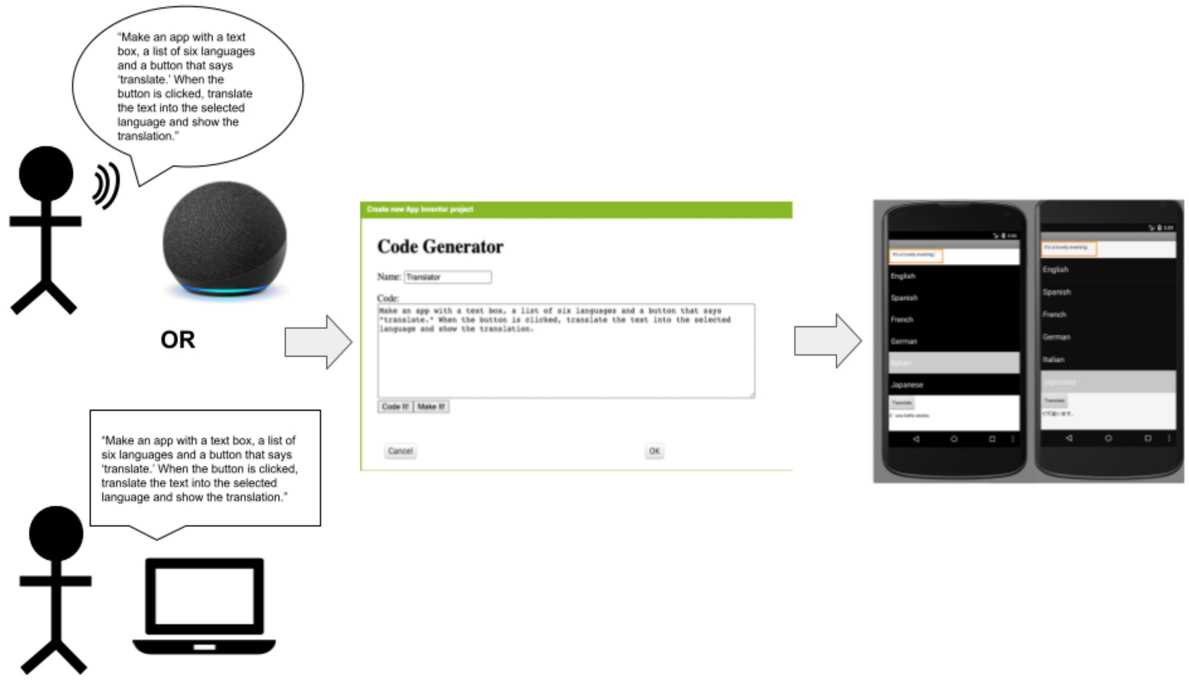
## 1 INTRODUCTION

We introduce Aptly, a platform that uses large language models to automatically generate mobile apps from written or spoken natural language descriptions. We encourage all people, especially young people, to identify problems in the real world and actively solve them by creating apps. Traditionally, however, a robust technical background has been necessary to code such apps, which discourages the process of app creation. The MIT App Inventor team has been developing an intuitive, visual programming environment that allows everyone to build fully functional apps [1]. Our research takes this simplification of app creation one step further. By developing Aptly, the team aims to democratize the process of creating apps, so that making apps is accessible to everyone, not just those with formal training in computer science.

This paper will first present how to use Aptly by introducing a simple example, followed by the technical implementation of Aptly along with a couple of preliminary results. Finally, we discuss Aptly's implications for the future of computational thinking education, challenging the traditional emphasis on coding and the need to manually translate ideas into code.

## 2 METHODOLOGY

### 2.1 How to use Aptly


"Make an app with a text box, a list of six languages and a button that says 'translate.' When the button is clicked, translate the text into the selected language and show the translation."
OR
"Make an app with a text box, a list of six languages and a button that says 'translate.' When the button is clicked, translate the text into the selected language and show the translation."
Create new App Inventor project
Code Generator
Code It! | Make It!
Cancel
OK
English
Spanish
French
German
Italian
Japanese
Translate

Name: Translator

Code:

```
Make an app with a text box, a list of six languages and a button that says
"translate." When the button is clicked, translate the text into the selected
language and show the translation.
```

Figure 1: An App Inventor translation app automatically generated with Aptly

MIT Aptly is a tool that generates Android and iPhone apps from natural language descriptions. For example, a user can type or speak the following to Aptly:

*Make an app with a text box, a list of six languages, and a button that says "translate." When the button is clicked, translate the text into the selected language and show the translation.*

The result is a working MIT App Inventor app with a text box for input and six buttons labeled "English", "Spanish", "French", "German", "Italian", and "Japanese" as shown above running on a phone [Figure 1]. Pressing one of the buttons translates the input to the corresponding language. This way, users can express an idea for an app they want and instantly view it on their phones[1].

In generating the app, Aptly makes inferences beyond the literal words and phrases in the text description. In this example, Aptly has independently chosen the six languages to be English, Spanish, French, German, Italian, and Japanese, and it has inferred that there should be a box where the user can provide the text to be translated. To accomplish this, Aptly draws upon a large body of code that serves as a context for the app generation.

In contrast to Aptly, even the simplest coding platforms featuring blocks-based interfaces, such as Scratch [2] or App Inventor [1], require technical work. Users of App Inventor, for example, would create the translation app by specifying components (elements such as buttons, labels, and text

[1] The video at https://www.youtube.com/watch?v=dT1i1MEr2Sg shows the creation of a similar translation app using Aptly, where the user speaks the description.

boxes) and their properties (attributes such as color and size) as shown in [Figure 2a], and assembling blocks as shown in [Figure 2b].

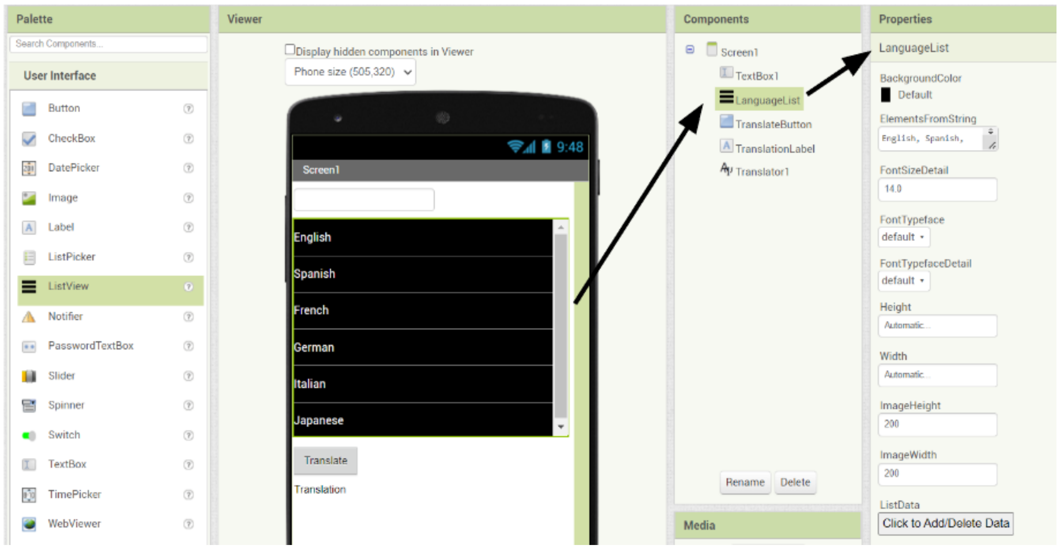
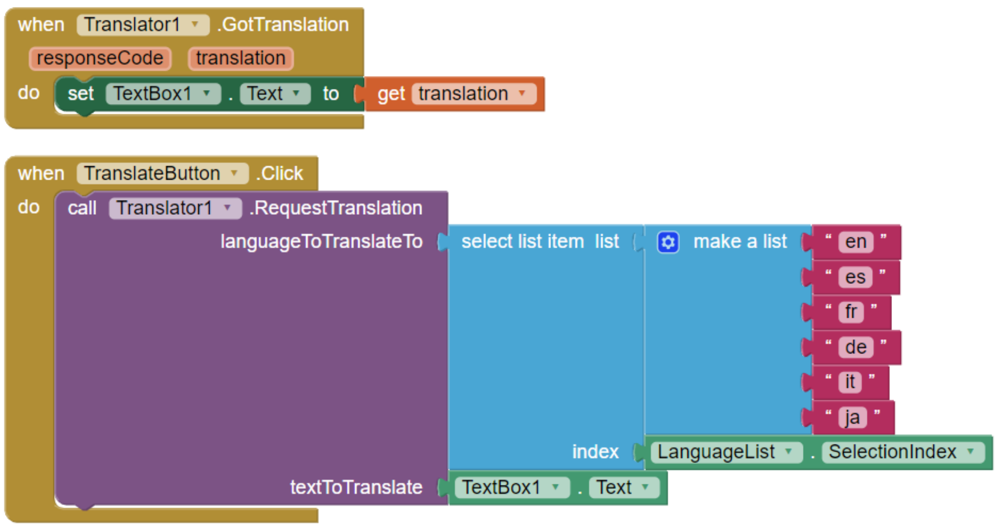


Figure 2a: Using App Inventor to specify components for the translation app. Aptly generates these automatically. This figure shows the component for the list of languages.



```
when Translator1 .GotTranslation
  responseCode  translation
do  set TextBox1 . Text to  get translation

when TranslateButton .Click
do  call Translator1 .RequestTranslation
        languageToTranslateTo  select list item list  make a list  " en "
                                                                  " es "
                                                                  " fr "
                                                                  " de "
                                                                  " it "
                                                                  " ja "
                                                     index  LanguageList . SelectionIndex
        textToTranslate  TextBox1 . Text
```

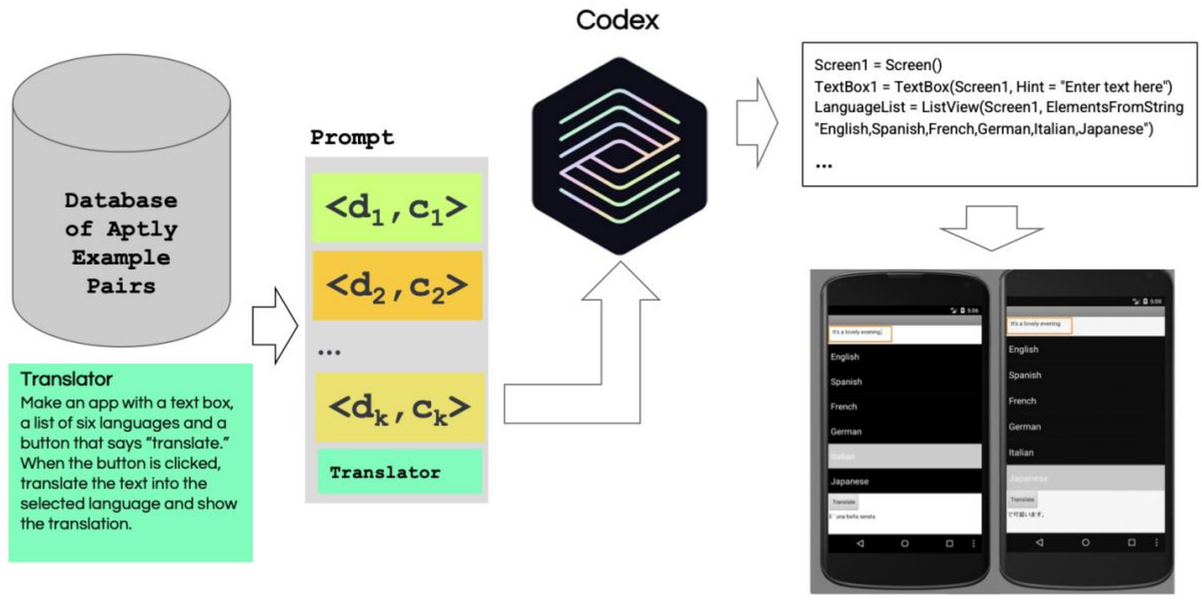
Figure 2b: App Inventor blocks for the translation app. Aptly generates these automatically.

With Aptly, the system generates the components, properties, and blocks automatically, then passes them on to App Inventor, which builds the working app.

## 2.2 How Aptly Works

Aptly is an application of machine learning research on *generative models*. Generative models are probabilistic models that describe how a dataset is generated which enables it to generate new data by sampling from these models. Specifically, we utilize a transformer-based [3] large language model to generate mobile apps. Recent demonstrations of large language models have been remarkably impressive, and this area is attracting a ferment of attention both for its research and commercial potential. Examples include systems that can create original text [4], computer code [5], music [6], and images [7]. The inputs to these generative models are called *prompts*. The performance of generative models is critically dependent on the quality of the prompts provided. Crafting appropriate prompts is known as *prompt engineering*, an active area of machine learning research [8]. One way of crafting a prompt is by providing a small number of examples of solved tasks as part of the input to the trained model, which is referred to as “few-shot” prompts [9]. The common interpretation of the few-shot prompting is that the model is “learning” the task during runtime from the few-shot examples.

The generative model Aptly uses is Open AI’s Codex [5], which generates computer code from natural language descriptions in several programming languages including Java, Javascript, Python, C, C#, and Swift. Codex itself is a derivative of OpenAI’s general generative model GPT-3 [9] which has been *fine-tuned* for computer code through training on more than 50 million Github repositories.



Translator
Make an app with a text box, a list of six languages and a button that says “translate.” When the button is clicked, translate the text into the selected language and show the translation.

```
Screen1 = Screen()
TextBox1 = TextBox(Screen1, Hint = "Enter text here")
LanguageList = ListView(Screen1, ElementsFromString
"English,Spanish,French,German,Italian,Japanese")
...
```

Figure 3 The whole process of how Aptly uses OpenAI’s Codex to generate a mobile app

[Figure 3] exhibits the general scheme of how we use Codex to generate a fully functional application. We automatically designed the prompt so that Codex would produce Aptly-Script, a similar language to Python specialized to have a one-to-one correspondence with App Inventor’s blocks. The design of Aptly-Script was motivated by the notion that proximity to Python syntax would allow the platform to take advantage of Codex’s Python training for basic language structures (e.g., that procedure arguments should be separated by commas and enclosed in balanced parentheses) while also maintaining a correspondence with App Inventor blocks. Here is the Aptly-Script generated for the example translation app above.

```
Screen1 = Screen()
TextBox1 = TextBox(Screen1, Hint = "Enter text here")
LanguageList = ListView(Screen1, ElementsFromString =
"English,Spanish,French,German,Italian,Japanese")
TranslateButton = Button(Screen1, Text = "Translate")
TranslationLabel = Label(Screen1)
Translator1 = Translator(Screen1)

when TranslateButton.Click():
        call Translator1.RequestTranslation (lists_select_item (["en", "es", "fr", "de", "it", "ja"],
LanguageList.Selection), TextBox1.Text)

when Translator1.GotTranslation(responseCode, translation):
        set TranslationLabel.Text = translation
```

When the user requests an app with their natural description, we synthesize a prompt which is a natural language description (denoted as $D$) of the desired app to be created, together with a set of example pairs, such as the following << $d_1, c_1$>> << $d_2, c_2$ >> ... << $d_k, c_k$>>, where $d_i$ is the description of application $i$, and $c_i$ is the Aptly-Script of application $i$. The example pairs come from a database of unique Aptly examples compiled by the team from apps created on the App Inventor platform. The example pairs are not expected to include the literal description $D$ to be processed nor the actual Aptly-Script to be generated which would make the generation task trivial. Instead, generative models use the provided examples to guide their processing in generating new original output.

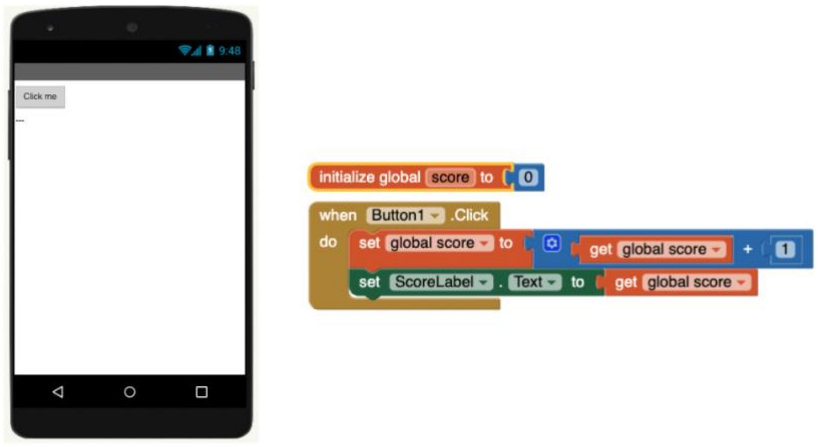
Aptly's method for providing good prompts to accompany a description $D$ is to provide a set of example application descriptions paired with appropriate Aptly-Script. We use *semantic similarity* to highlight the "appropriateness" of code and description. The method relies on having a way to automatically measure the (semantic) similarity of text descriptions and the program. For computing the similarity of the text description and program, we use *embeddings* [10] of data elements. Embeddings are numerical representations of concepts converted to number sequences. In our scenario, an embedding represents the semantic meaning of a natural language description or code [11]. To measure the similarity of two items, one embeds them in the same vector space and takes the distance between the two embedding vectors as a measure of similarity (Small distance implies high similarity while large distance implies low similarity). We use *cosine distance*, which reflects the angle between vectors, to compute the distance between two vectors.

We use OpenAI's Babbage engine model [11] to obtain the embeddings of the description and code. These Neural Network models were trained using a technique called *Contrastive Pre-Training* [12]. Contrastive pre-training is a kind of clustering training technique where in the vector space the predefined positive examples (i.e. matching text and code) move the items toward each other and the negative examples (i.e. contrasting text and code) move the items away from each other. Using this model will allow us to select the most relevant example pair to the user's requested description.

We feed the synthesized prompt into Codex, which outputs the Aptly-Script corresponding to the user's requested app description. We can then convert the generated Aptly-Script into App Inventor blocks to generate a fully functional application (Figure 3)

## 3 PRELIMINARY RESULTS

Here we show a couple of working examples. We picked examples that are potential apps that children would create. App creation consists of creating the UI design and implementation of the functionality.



```
initialize global score to 0
when Button1 .Click
do  set global score to  get global score + 1
    set ScoreLabel . Text to  get global score
```

Figure 4 Working Example 1

If a user types or speaks the following to Aptly:

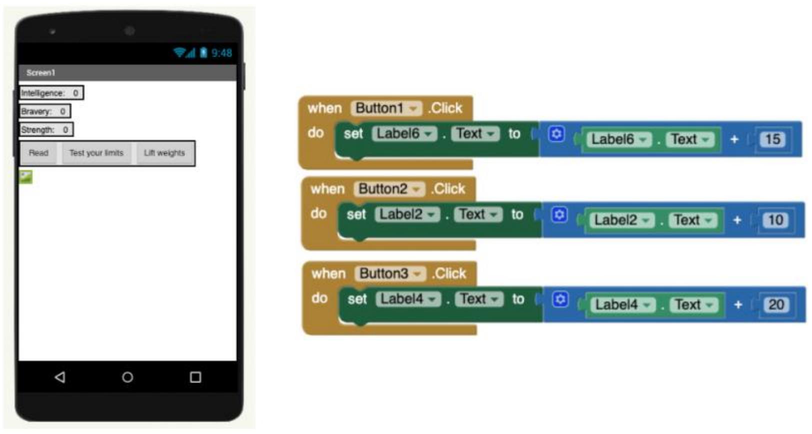
> *Make a game that has a button in the middle of the screen. When a user clicks the button, increment the score by 1*

The result is a working MIT App Inventor app with a button that says Click me, as shown in [Figure 4]. When a user presses the button will increase the global variable score by 1.

This app can be assembled by first creating a button and incrementing the score by 1 when the button is pressed. As we can see, the basic functionality of the requested app is correctly implemented. However, it failed to properly place the button in the middle of the screen.

If a user types or speaks the following to Aptly:

> *Create a character that has attributes: intelligence, bravery, and strength. If the reader clicks the 'read' button, increment intelligence by 10. If the reader clicks the 'Test your limits' button, increase bravery by 20. If the reader clicks the 'lift weights' button, increase strength by 15.*



```
when Button1 .Click
do set Label6 . Text to Label6 . Text + 15

when Button2 .Click
do set Label2 . Text to Label2 . Text + 10

when Button3 .Click
do set Label4 . Text to Label4 . Text + 20
```

Figure 5 Working example 2

The result is a working MIT App Inventor app with three buttons that says “Read”, “Test your limits”, and “Lift weights” respectively, as shown in [Figure 5]. Pressing the button “Read” will increase the “Intelligence” label text by 15. Pressing the button “Test your limits” will increase the “Bravery” label text by 10. Pressing the button “Lift weights” will increase the “Strength” label text by 20.

In comparison to the previous app, there are multiple buttons to create, and each button has different functionality and its respective name. On top of that, it requires you to show an image of a character. We still observe that the basic functionality of the apps is correctly created. However, we see that the image of the character is missing (below the buttons). Certain apps will require images and sounds which we call "assets". How to correctly insert assets is another challenge that our team plan to solve.

## 4 CONCLUSIONS

### 4.1 Aptly’s challenge to computational thinking education

The role of programming in computational thinking [13] remains a topic of active discussion among computer education curriculum developers. Some developers regard practice with coding as fundamental to the curriculum. Others consider coding to be marginal to computational thinking, preferring to focus on conceptual ideas about computation, independent of how they are expressed in computer languages.

Aptly aligns with the second view. It's closely related to today's burgeoning "no-code" movement [14] that seeks to lessen the need for expert programmers in building industrial applications. No-code aims to make software development accessible to everyone, removing barriers so that people without extensive training in coding can bring ideas to life. Aptly offers the same approach in K-12 education, where the emerging software tools could eliminate much of the need for learners to deal with code.

Much of today's activity in computational thinking traces back to the 1970's MIT work of Seymour Papert, where a key principle was that children can practice formal thinking by expressing ideas in a specially designed computer language (Logo [15]). What becomes of this approach when Aptly and its successors let the computer automatically recast ordinary language as code? Is the computer now doing the computational thinking on its own? Has the need to teach computational thinking been obviated?

Aptly points to a new perspective on computational thinking and raises new questions: Does the tool diminish the need for humans to engage in computational thinking, and does it signal an end to some of the core skills currently associated with computational thinking? Or will it inspire more people to dig deeper into the process of creating programs and let them address new computing possibilities that transcend coding? Or could it represent a combination of both?

More research on learning outcomes both with beginners and experienced coders is needed. There are new educational possibilities to explore. For example, learners with some experience can get started on an app using Aptly and then edit the Aptly-created blocks to add features themselves. People who want to practice debugging can describe an intended app, have Aptly created one that mostly matches the description, and then practice finding any bugs that cause the Aptly-generated app to not function as intended.

Aptly could provide evidence of surprising features of contemporary machine learning as it applies to computer science education, such as the idea that computers can generalize beyond the specific instructions they have been given.

## 4.2 Aptly's future

Aptly is currently a limited prototype. It does not support every App Inventor feature and its Codex-based training uses only a few examples. Nevertheless, today's explosive activity in large-language models and no-code platforms gives confidence that the performance of Aptly and similar tools will improve and that their use in K-12 education will grow.

What will be the impact of tools like Aptly in computing education? Consider an analogy with the use of calculators in mathematics education. The introduction of calculators in the 1970s sparked debates about their use in education that are still ongoing after 50 years. Some educators resist calculators, worrying that they would become crutches that undermine the learning of arithmetic skills. Others embrace the opportunity for calculators to be springboards that let students move beyond arithmetic to access higher-level mathematics concepts.

*Will Aptly be a crutch or a springboard?*

We can expect similar debates in computational thinking education around coding skills. This is our vision for Aptly: Children and adults alike will be able to create meaningful apps without prior experience in programming. Creating apps — once inaccessible to non-programmers — will require less expertise, allowing people's ideas to blossom. That's both a challenge and an opportunity for practitioners of educational computing.

## ACKNOWLEDGEMENTS

We thank the App Inventor team for supporting our research. We also thank Juston Forte from OpenAI for giving us insights about using embeddings to optimize performance.

## REFERENCES

[1] Wolber, David, Hal Abelson, Ellen Spertus, and Liz Looney. *App inventor*. " O'Reilly Media, Inc.", 2011.

[2] Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner et al. "Scratch: programming for all." *Communications of the ACM* 52, no. 11 (2009): 60-67.

[3] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[4] Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan et al. "Language models are few-shot learners." *Advances in neural information processing systems* 33 (2020): 1877-1901.

[5] Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards et al. "Evaluating large language models trained on code." *arXiv preprint arXiv:2107.03374* (2021).

[6] Krol, Stephen James, Maria Teresa Llano, and Jon McCormack. "Towards the Generation of Musical Explanations with GPT-3." In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pp. 131-147. Springer, Cham, 2022.

[7] Ramesh, Aditya, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. "Zero-shot text-to-image generation." In *International Conference on Machine Learning*, pp. 8821-8831. PMLR, 2021.

[8] Reynolds, Laria, and Kyle McDonell. "Prompt programming for large language models: Beyond the few-shot paradigm." In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1-7. 2021.

[9] Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan et al. "Language models are few-shot learners." *Advances in neural information processing systems* 33 (2020): 1877-1901.

[10] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532-1543. 2014.

[11] Chen, Zimin, and Martin Monperrus. "A literature study of embeddings on source code." *arXiv preprint arXiv:1904.03061* (2019).

[12] Neelakantan, Arvind, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan et al. “Text and code embeddings by contrastive pre-trainging.” arXiv preprint arXiv:2201.10005 (2022).

[13] Wing, Jeannette M. "Computational thinking." *Communications of the ACM* 49, no. 3 (2006): 33-35.

[14] Satell, Greg (21 April 2018). "The Future of Software Is No-Code". *www.inc.com*. Retrieved 20 August 2018.

[15] Abelson, Hal, Nat Goodman, and Lee Rudolph. "Logo manual." (1974).