Can Large Language Models Reduce the Barriers to Entry for High School Robotics?

William Xie[1], Brian Plancher[2]

Abstract— **In this study we will investigate whether we can reduce the barriers to entry for high school robotics through the use of code generation models derived from large language models (LLMs). As such, we aim to raise the abstraction barrier for the development of artificial intelligence algorithms needed to program and control the Romi Robot used in the FIRST Robotics Competition (FRC). To do so we develop a web interface that helps automate the prompt-engineer step and allows students to easily incorporate OpenAI Codex into their workflows. To evaluate the impact of our approach, we will survey students to understand their overall experience and their satisfaction with, and perceived usefulness of, this technology. Additionally we will survey FRC community members to understand the community perception of the importance and equity of programming education within the context of high school robotics. We hope this study helps chart a path towards reduced coding prerequisites for high school robotics.**

I. INTRODUCTION

The FIRST Robotics Competition (FRC) is a large scale high school robotics competition in which thousands of teams from around the world compete to build and control a 120 lb robot for a novel game challenge each year [1]. This requires the team to explore and learn interdisciplinary topics with a high floor ranging from computer science, to control systems, to electronics, to mechanical design. While some teams are well funded and supported by large teams of professional educators or engineers with ample time to volunteer, many only have the support of one or two limited volunteers. As such, there are currently vast resource and learning inequities in FRC. These issues are particularly stark with regards to programming as teams often have to not only leverage domain-specific and custom vendor software libraries, but also develop artificial intelligence algorithms to achieve desired tasks, while overcoming the particular characteristics of their novel physical robot.

At the same time, recent work has shown that code generation models, derived from large language models (LLMs) and leveraging natural language descriptions like Github CoPilot or OpenAI models (Codex, GPT-3, GPT-4, ChatGPT) can be very successful for assisting novice programmers in university introductory courses [2], [3]. However, these tools have been underexplored in lower-resource domains and younger student audiences. And, while the introductory programming sequence has a large web corpora of data on which these models have been trained, the FRC robotics context presents a limited web corpus with new target tasks each year.

Therefore, to help brige this resource gap, we aim explore the following research question:

> *Can code generation models reduce the software engineering barriers to entry for high school robotics?*

To do so, we will develop a web interface that automates much of the prompt-engineering phase of effective LLM usage to allow all students, regardless of mentor resources, to quickly and easily integrate Codex into their workflows. We will then have students use Codex to help them develop software implementations of artificial intelligence algorithms needed to program and control the Romi Robot used in the FIRST Robotics Competition (FRC).

To evaluate the impact of our approach, we will survey students to understand their overall experience and their satisfaction with, and perceived usefulness of, this technology. Additionally we will survey FRC community members and alumni to understand the community perception of the importance and equity of programming education within the context of high school robotics.

II. RELATED WORK

A. LLMs for Programming Education

Code generation models are an emergent technology in programming education with both potential to assist learning by explaining code, expediting syntax, and debugging errors and potential to detract from learning by fostering overreliance on code generation, providing incorrect or suboptimal guidance, and leading students to academic misconduct.

In their position paper, Becker et al. [4] urged educators to proactively identify opportunities and challenges presented by these tools to guide their development and application in teaching. Kazemitabaar et al. [2] found that OpenAI Codex improved code-authoring performance without decreasing manual code-modification skills among young Scratch learners. Prather et al. [3] observed that novice programmers were both wary of and optimistic about GitHub Copilot, with many of them being confused by generated code and falling into patterns of either over-reliance or cyclic hesitation.

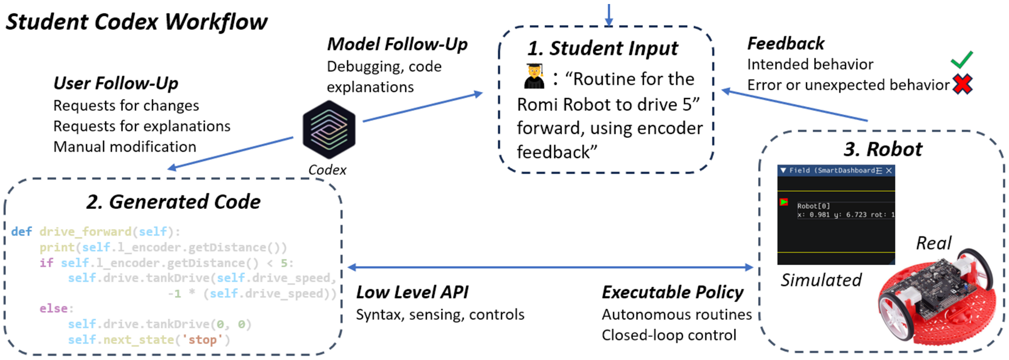
B. LLMs for Robot Learning

In robotics, code generation models have synergized with natural language for the task and motion planning problem. Given natural language instructions as input, code generation models are able to both formulate high-level plans and decompose them into sequences of programmatic actions. Yu et al. [5] used OpenAI GPT-4 to atomicize robot motion plans inputted as natural language into valid MuJoCo MPC reward parameters, bridging the gap between high-level language

---

[1] William Xie is with University of Colorado at Boulder, Boulder, CO. `wixi6454@colorado.edu`

[2] Brian Plancher is with Barnard College, Columbia University, New York, NY. `bplancher.barnard.edu`



```python
def drive_forward(self):
    print(self.l_encoder.getDistance())
    if self.l_encoder.getDistance() < 5:
        self.drive.tankDrive(self.drive_speed,
                             -1 * (self.drive_speed))
    else:
        self.drive.tankDrive(0, 0)
        self.next_state('stop')
```

Fig. 1. Students will interact with a custom instance of Codex through a web interface to generate and debug code for deployment on a physical robot.

instructions and low-level robot actions. Code as Policies [6] used Codex to generate executable robot policy code from natural language commands, capable of spatial-geometric reasoning, generalizing to new instructions, and prescribing specific values in ambiguous instructions.

III. STUDY DESIGN

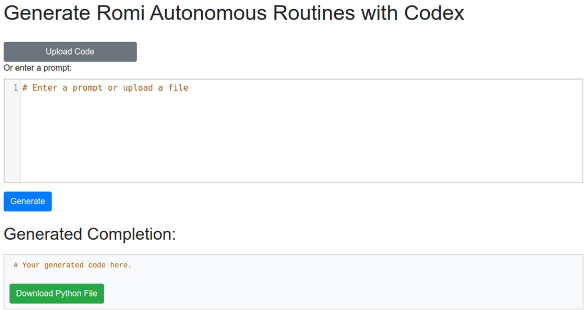
Our study will be centered on a group of high school aged students participating in FRC during the 2023-2024 academic year. We will provide them with access to a web interface that automates much of the prompt-engineering phase of effective LLM usage, as well as provide introductory education on programming, Codex, and robotics. We will then ask them to leverage our interface to assist in the development of artificial intelligence algorithms needed to complete FRC tasks.

A. Prompt-Engineering Interface Design

To improve out-of-the-box code policy generation and remove friction in student interaction for simple commands, an instance of Codex (`temperature=1, max_tokens=256`) running in an easy-to-use web interface is provided to students. This interface, automatically appends a large prompt detailing Romi-specific library and sensor usage to student questions. Importantly, this prompt includes a few carefully chosen exemplars of language interaction to policy code generation (See Appendix B). This is critical as LLMs need to be provided with examples of natural language commands and corresponding policy code (few-shot prompting), in order to effective receive new commands and recompose API calls to generate new policy code [6]. To make it easier for students to interact with Codex, we developed a web-hosted question-and-answering interface (see Figure 2), a demo version of which can be found at `codex-for-romi.williamxie.nyc`.

B. Student Workflow

Students will primarily employ Codex to develop the artificial intelligence algorithms used to control the Romi robot. Students will initially interact with the Romi robot and its components via an FRC-specific Python simulator, RobotPy Robot Simulator, which will also be used to transition to real robot deployments. Students will interact with Codex through our web interface which will allow them to gain feedback on their code, develop new code, and integrate code snippits into fully functioning software systems (See Figure 1), without needing to understand all of the various low-level APIs needed to leverage the Romi robot.



```
# Enter a prompt or upload a file
```

```
# Your generated code here.
```

Fig. 2. A screenshot of web interface which allows students to interact with Codex through a question-and-answering interface. See Appendix A for an code generation example.

C. Student and Community Surveys

Throughout the duration of the FRC competition season students will take surveys on their experience. These survey questions fall into three categories: self-confidence in using Codex, emotional response to using Codex, and assessment of Codex usage, reliability, and understanding.

To gather robotics community input on programming education, survey participants will be recruited from ChiefDelphi, an online forum for FRC frequented by adult mentors and alumni. Participants will be asked questions relating to their team's programming education, as well as their opinion on both the impact of, and equitable access to, programming education and mentorship for FRC students.

IV. CONCLUSION AND NEXT STEPS

In this paper we describe the design of a study to evaluate the potential for code generation models to reduce the barriers to entry for high school robotics. In the forthcoming academic year we will leverage this approach with a group of students participating in FRC[1]. We will also continue to develop our approach to both further reduce prompt complexity as well as better automate the process of connecting the generated code to the simulated and physical robot. We hope this effort is an important first step in lowering the software engineering barrier of entry to robotics.

---

[1] The exact student team(s) are being actively recruited and IRB approval is in-progress. We aim to recruit more than one team for scalability analysis.

REFERENCES

[1] A. Melchior, F. Cohen, T. Cutter, T. Leavitt, and N. Manchester, "More than robots: An evaluation of the first robotics competition participant and institutional impacts," *Heller School for Social Policy and Management, Brandeis University*, 2005.

[2] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of ai code generators on supporting novice learners in introductory programming," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, Apr 2023, p. 1–23, arXiv:2302.07427 [cs]. [Online]. Available: http://arxiv.org/abs/2302.07427

[3] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos, ""it's weird that it knows what i want": Usability and interactions with copilot for novice programmers," no. arXiv:2304.02491, Apr 2023, arXiv:2304.02491 [cs]. [Online]. Available: http://arxiv.org/abs/2304.02491

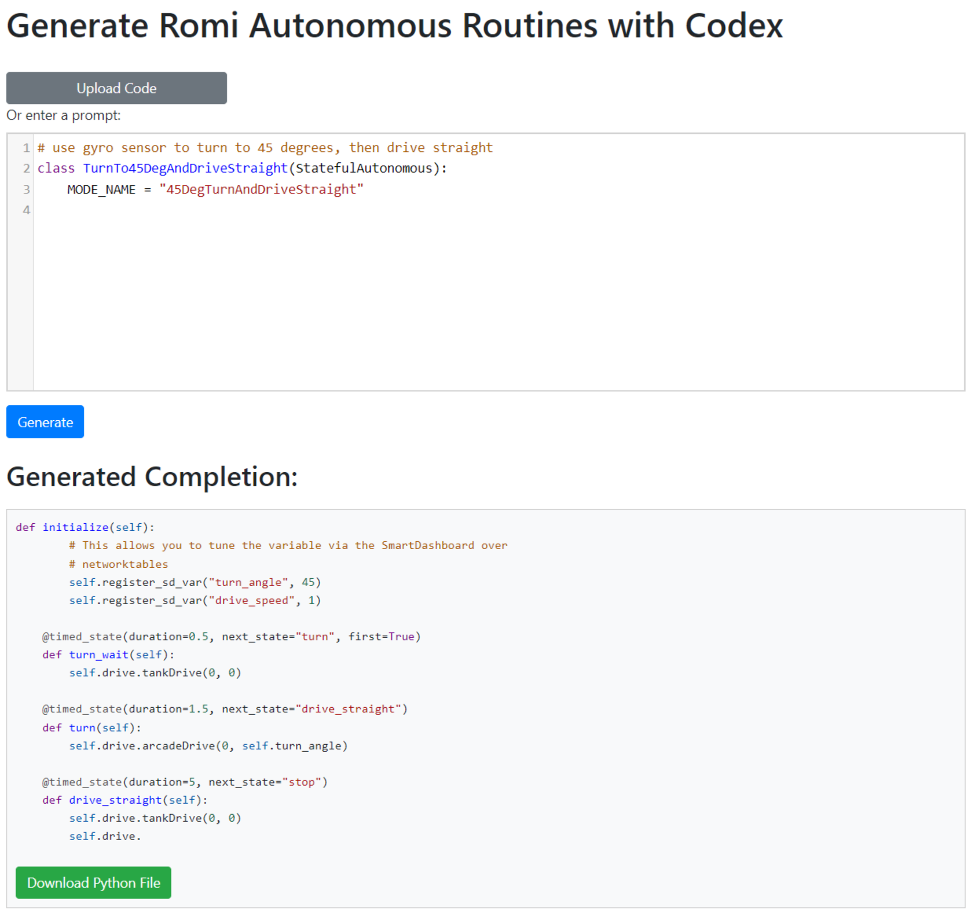
[4] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard – or at least it used to be: Educational opportunities and challenges of ai code generation," Dec 2022. [Online]. Available: https://arxiv.org/abs/2212.01020v1

[5] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," no. arXiv:2306.08647, Jun 2023, arXiv:2306.08647 [cs]. [Online]. Available: http://arxiv.org/abs/2306.08647

[6] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," no. arXiv:2209.07753, May 2023, arXiv:2209.07753 [cs]. [Online]. Available: http://arxiv.org/abs/2209.07753

APPENDIX

A. *Our Web Interface with Example Generated Code*



Generate Romi Autonomous Routines with Codex

Or enter a prompt:

```
# use gyro sensor to turn to 45 degrees, then drive straight
class TurnTo45DegAndDriveStraight(StatefulAutonomous):
    MODE_NAME = "45DegTurnAndDriveStraight"

```

Generated Completion:

```
def initialize(self):
        # This allows you to tune the variable via the SmartDashboard over
        # networktables
        self.register_sd_var("turn_angle", 45)
        self.register_sd_var("drive_speed", 1)

    @timed_state(duration=0.5, next_state="turn", first=True)
    def turn_wait(self):
        self.drive.tankDrive(0, 0)

    @timed_state(duration=1.5, next_state="drive_straight")
    def turn(self):
        self.drive.arcadeDrive(0, self.turn_angle)

    @timed_state(duration=5, next_state="stop")
    def drive_straight(self):
        self.drive.tankDrive(0, 0)
        self.drive.
```

Fig. 3. A screenshot of web interface with generated code from a custom Codex instance leveraging in-context learning on few-shot exemplars.

B. Romi Robot Default Codex Prompt Appended to Student Prompts

```
# Robot Definition Code
#!/usr/bin/env python3

import wpilib
import wpilib.drive
import romi
import math

from robotpy_ext.autonomous import AutonomousModeSelector
class MyRobot(wpilib.TimedRobot):
    """
    This shows using the AutonomousModeSelector to automatically choose
    autonomous modes.

    If you find this useful, you may want to consider using the Magicbot
    framework, as it already has this integrated into it.
    """
    kCountsPerRevolution = 1440.0
    kWheelDiameterInch = 2.75591


    def robotInit(self):
        self.lstick = wpilib.Joystick(0)
        self.rstick = wpilib.Joystick(1)

        # Simple two wheel drive
        self.l_motor = wpilib.Talon(0)
        self.r_motor = wpilib.Talon(1)

        # wheel encoders
        self.l_encoder = wpilib.Encoder(4, 5)
        self.r_encoder = wpilib.Encoder(6, 7)

        # Set up the BuiltInAccelerometer
        self.accelerometer = wpilib.BuiltInAccelerometer()

        self.drive = wpilib.drive.DifferentialDrive(self.l_motor, self.r_motor)

        # Position gets automatically updated as robot moves
        self.gyro = wpilib.AnalogGyro(1)
        # Set up the RomiGyro
        # self.gyro = romi.RomiGyro()

        # unused
        self.motor = wpilib.Talon(2)
        self.limit1 = wpilib.DigitalInput(1)
        self.limit2 = wpilib.DigitalInput(2)
        self.position = wpilib.AnalogInput(2)

        # Items in this dictionary are available in your autonomous mode
        # as attributes on your autonomous object
        self.components = {"drive": self.drive, "gyro": self.gyro, "motor": self.motor,
                           "l_motor": self.l_motor, "r_motor": self.r_motor,
                           "l_encoder": self.l_encoder, "r_encoder": self.r_encoder,
                           "limit1": self.limit1, "limit2": self.limit2,
```

```
                     "position": self.position }

        # * The first argument is the name of the package that your autonomous
        #   modes are located in
        # * The second argument is passed to each StatefulAutonomous when they
        #   start up
        self.automodes = AutonomousModeSelector("autonomous", self.components)

        # Use inches as unit for encoder distances
        self.l_encoder.setDistancePerPulse(
            (math.pi * self.kWheelDiameterInch) / self.kCountsPerRevolution
        )
        self.r_encoder.setDistancePerPulse(
            (math.pi * self.kWheelDiameterInch) / self.kCountsPerRevolution
        )
        self.resetEncoders()
    def resetEncoders(self) -> None:
        """Resets the drive encoders to currently read a position of 0."""
        self.l_encoder.reset()
        self.r_encoder.reset()

    def getl_encoderCount(self) -> int:
        return self.l_encoder.get()

    def getr_encoderCount(self) -> int:
        return self.r_encoder.get()

    def getLeftDistanceInch(self) -> float:
        return self.l_encoder.getDistance()

    def getRightDistanceInch(self) -> float:
        return self.r_encoder.getDistance()

    def getAverageDistanceInch(self) -> float:
        """Gets the average distance of the TWO encoders."""
        return (self.getLeftDistanceInch() + self.getRightDistanceInch()) / 2.0

    def getAccelX(self) -> float:
        """The acceleration in the X-axis.

        :returns: The acceleration of the Romi along the X-axis in Gs
        """
        return self.accelerometer.getX()

    def getAccelY(self) -> float:
        """The acceleration in the Y-axis.

        :returns: The acceleration of the Romi along the Y-axis in Gs
        """
        return self.accelerometer.getY()

    def getAccelZ(self) -> float:
        """The acceleration in the Z-axis.

        :returns: The acceleration of the Romi along the Z-axis in Gs
        """
        return self.accelerometer.getZ()
```

```
    def getGyroAngleX(self) -> float:
        """Current angle of the Romi around the X-axis.

        :returns: The current angle of the Romi in degrees
        """
        return self.gyro.getAngleX()

    def getGyroAngleY(self) -> float:
        """Current angle of the Romi around the Y-axis.

        :returns: The current angle of the Romi in degrees
        """
        return self.gyro.getAngleY()

    def getGyroAngleZ(self) -> float:
        """Current angle of the Romi around the Z-axis.

        :returns: The current angle of the Romi in degrees
        """
        return self.gyro.getAngleZ()

    def resetGyro(self) -> None:
        """Reset the gyro"""
        self.gyro.reset()

    def autonomousInit(self):
        self.drive.setSafetyEnabled(True)
        self.automodes.start()

    def autonomousPeriodic(self):
        self.automodes.periodic()

    def disabledInit(self):
        self.automodes.disable()

    def teleopPeriodic(self):
        """Called when operation control mode is enabled"""

        self.drive.arcadeDrive(self.lstick.getX(), self.lstick.getY())

        # Move a motor with a Joystick
        y = self.rstick.getY()

        # stop movement backwards when 1 is on
        if self.limit1.get():
            y = max(0, y)

        # stop movement forwards when 2 is on
        if self.limit2.get():
            y = min(0, y)

        self.motor.set(y)

# Robot Autonomous Code
from robotpy_ext.autonomous import StatefulAutonomous, state, timed_state
class DriveBackwards(StatefulAutonomous):
```

```
    MODE_NAME = "Drive Backwards"

    def initialize(self):
        # This allows you to tune the variable via the SmartDashboard over
        # networktables
        self.register_sd_var("drive_speed", -1)

    @timed_state(duration=0.5, next_state="drive_backwards", first=True)
    def drive_wait(self):
        self.drive.tankDrive(0, 0)

    @timed_state(duration=5, next_state="stop")
    def drive_backwards(self):
        self.drive.tankDrive(self.drive_speed, -1 * (self.drive_speed))

    @state()  # Remove or modify this to add additional states to this class.
    def stop(self):
        self.drive.tankDrive(0, 0)

# User Request: autonomous routine to drive forward for 5 seconds
class DriveForward(StatefulAutonomous):
    MODE_NAME = "Drive Forward"

    def initialize(self):
        # This allows you to tune the variable via the SmartDashboard over
        # networktables
        self.register_sd_var("drive_speed", 1)

    @timed_state(duration=0.5, next_state="drive_forward", first=True)
    def drive_wait(self):
        self.drive.tankDrive(0, 0)

    @timed_state(duration=5, next_state="stop")
    def drive_forward(self):
        self.drive.tankDrive(self.drive_speed, -1 * (self.drive_speed))

    @state()  # Remove or modify this to add additional states to this class.
    def stop(self):
        self.drive.tankDrive(0, 0)

# User Request: autonomous routine to drive five inches, using encoder feedback
class DriveFiveInches(StatefulAutonomous):
    MODE_NAME = "Drive 5 Inches"
    def initialize(self):
        # This allows you to tune the variable via the SmartDashboard over
        # networktables
        self.register_sd_var(’drive_speed’, 1)

    @timed_state(duration=0.5, next_state="drive_forward", first=True)
    def drive_wait(self):
        self.l_encoder.reset()
        self.r_encoder.reset()
        self.drive.tankDrive(0, 0)

    @state()
    def drive_forward(self):
        print(self.l_encoder.getDistance())
```

```
    if self.l_encoder.getDistance() < 5:
        # motor speeds chosen through experimentation
        self.drive.tankDrive(self.drive_speed, -1 * (self.drive_speed))
    else:
        self.drive.tankDrive(0, 0)
        self.next_state('stop')

@state()  # Remove or modify this to add additional states to this class.
def stop(self):
    self.drive.tankDrive(0, 0)
```