Automation of NC Programming with Artificial Intelligence

by

Michael Lunny

Submitted to the MIT Sloan School of Management

and

MIT Department of Aeronautics and Astronautics

in partial fulfillment of the requirements for the degrees of

Master of Business Administration

and

Master of Science in Aeronautics and Astronautics Engineering

in conjunction with the Leaders for Global Operations program

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

© Michael Lunny, 2022. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author .....

Michael Lunny

Sloan School of Management and Department of Aeronautics and Astronautics

May 6, 2022

Certified by .....

Paulo Lozano, Thesis Supervisor

Professor of Aeronautics and Astronautics

Certified by .....

Daniel Freund, Thesis Supervisor

Assistant Professor of Operations Management

Accepted by .....

Jonathan How, Chair, Graduate Program Committee

Professor of Aeronautics and Astronautics

Accepted by .....

Maura Herson, Assistant Dean

MBA Program, Sloan School of Management

Automation of NC Programming with Artificial Intelligence

by

Michael Lunny

Submitted to the MIT Sloan School of Management
and
MIT Department of Aeronautics and Astronautics
on May 6, 2022, in partial fulfillment of the
requirements for the degrees of
Master of Business Administration
and
Master of Science in Aeronautics and Astronautics Engineering
in conjunction with the Leaders for Global Operations program

Abstract

With the advent of artificial intelligence (AI) in business operations of various industries in recent decades, manufacturing firms are embracing intelligent, data-driven methods of making their processes more efficient. In particular, AI-driven automation of computer numerically controlled (CNC) programming, the process by which cutting tool and operation parameters governing CNC machines are determined, has potential to yield dramatic benefits to machining companies. Within the context of Midwest-based machining firm Orizon, two approaches to programming automation were developed. Geometry Rule-based Automation of Programming (GRAP) is a rule-based system with the ability to recognize hole and pocket features and automatically create an associated program, albeit suboptimal. Deep Learning for Automated Tool Selection (DLATS) is a machine learning algorithm with the ability to select the appropriate cutting tool for a hole drilling process with 32% accuracy, which is over 300 times better than random selection. Motivation, results, and implementation findings for both GRAP and DLATS are presented.

Thesis Supervisor: Paulo Lozano
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Daniel Freund
Title: Assistant Professor of Operations Management

Acknowledgments

The following people were integral to the completion of this research and thesis: Daniel Freund and Paulo Lozano, my advisors who provided insightful feedback and guidance as my ideas were developed and documented; Jeff Birenbaum, Jessica Wu, Danny Davis, my AIP supervisor (Jeff) and mentors (Jessica and Danny) who challenged my assertions and improved my impact on bottom-line value through their experience with industrial firms; Josh Fink, Charlie Newell, Henry Newell, Gary Appleton, Lance Sanders, Ryon Groff, Brian Holloway, Travis Wagner, and Jason Hensley, my Orizon supervisor (Josh), executive leaders (Charlie and Henry) and colleagues (rest) that enabled me to conduct my research at their company, embraced my outlandish approaches to NC programming, and showed me how a world-class manufacturing facility machines and assembles hardware.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Large aerospace and defense systems companies (e.g., Lockheed Martin, Northrop Grumman, Spirit, etc.) rely heavily on suppliers to produce components and assemblies which are integrated into an end product. For example, the Lockheed Martin F35, a military combat aircraft, is built with parts sourced from various suppliers [1]; one key supplier to the F35 platform is Orizon AeroStructures, a manufacturing company that fabricates and assembles airframes. While under contract, Orizon is obligated to supply airframe assemblies at a rate commensurate to Lockheed Martin's F35 production rate.

In order to optimize the production process to supply assemblies at the appropriate rate, Orizon must invest in its upfront development with non-recurring engineering (NRE) process costs prior to production. This process, referred to as onboarding, involves engineering investment to reduce the cycle time for parts in production—specifically, Orizon fabricates large metallic parts using computer numeric controlled (CNC) machines. Onboarding includes developing a CNC program (i.e., line-by-line instructions for tool choices, tooling paths, machine parameters, etc.) to operate the machine, designing workholding fixturing to secure the part while on the machine, and testing the program and workholding to ensure that, in a production mode, the process can meet cycle time and quality requirements.

A natural trade-off exists between upfront investment in an optimized machining process and rapid development of a suboptimal process: in contrast to Orizon, a manufacturing firm can immediately bring a part into production without an optimized machining process. Ultimately, this rapid development approach results in longer cycle times, failure to meet quality requirements, and reliance on skilled machinists to make real-time process adjustments. This approach may be justifiable if investment in the development process meaningfully reduces profit margins for low-volume parts. Orizon typically agrees to high-volume contracts and consequently maximizes profit margins with notable investments in NRE.

Perhaps the most significant NRE cost is CNC programming (hereafter referred to as programming or NC programming). On a part number basis, programming may take up to 2,000 hours, equating to a cost of roughly $100k per part number. While programming cost negatively impacts Orizon's achievable profit margin, the large amount of labor hours required to program a part also prohibits optimal resource allocation and thus reduces velocity of parts to customer delivery. Were Orizon able to reduce the average number of programming hours per part, the company's programming resources could be more effectively used to onboard more parts per programming hour; such an improvement would justify a pursuit to capture a larger, more diverse customer portfolio and invest in a proportional number of machining assets to achieve revenue and earnings growth. As a result, automation of the programming process is a key business need for Orizon.

1.1.1 Overview of Contents

This study presents methods of automating programming with the motivation of reducing labor costs and improving product velocity through the onboarding process within the context of Orizon. Section 1.2 introduces basic considerations and definitions for CNC machining and the programming process; Section 1.3 discusses the onboarding process and programming as an integral part of that process; Section 1.4 presents financial data motivating the desire to reduce onboarding and programming time; and Section 1.5 contains contemporary industry practices and academic

research related to programming automation.

Chapter 2 presents Geometry Rule-Based Automation of Programming (GRAP), a nascent artificial intelligence (AI) system for automatically creating programs; Chapter 3 introduces Deep Learning for Automated Tool Selection (DLATS), a machine learning (ML) algorithm for automatically selecting cutting tools (an integral part of the programming process, see Section 1.2); Chapter 4 compares GRAP and DLATS; and Chapter 5 discusses implications of the results herein on future developments towards programming automation.

1.2 CNC Machining and Programming

Orizon is a premier manufacturer of structural assemblies composing various aerospace platforms, but most notably aircraft. These assemblies are typically made of large, metallic components that maintain much of the structural integrity of the fully-assembled aerospace system. To manufacture these assemblies, Orizon first machines (i.e., cuts) stock material on a CNC machine to produce parts with the desired geometrical tolerance requirements specified by their customers. These parts vary in size, complexity, and material type, but typically require a few hours of machining time. Many of the parts Orizon machines are highly complex, requiring a substantial engineering effort to program the machine to cut the part within quality requirements (i.e., geometrical tolerances) and acceptable cycle times. More information describing these processes is contained in Sections 1.2.1 and 1.3.

Parts are machined at one of three machine shops then delivered to a central facility responsible for processing and building the parts into the assemblies required by the customers. Final assemblies are then shipped to the customers for integration into the target higher-level assembly.

1.2.1 Computer-aided Machining Workflow Definitions

Computerization of manufacturing workflows since the 1950s has largely defined the state of machining processes today [2, 3, 4]. Computer-aided design (CAD) soft-

ware allows engineers to develop a 3-D digital representation of a desired part to be fabricated. A common CAD file type is the STEP, or .stp, file, which is a method to represent and exchange product manufacturing information governed by standard ISO 10303 [5]. Such CAD files are key inputs to the computer-aided manufacturing (CAM) process, which is the process to define and execute manufacturing instructions to fabricate a given part. CAM software allows users to create, test, and verify CNC programs, which define the step-by-step commands for CNC machines to fabricate a desired end part. CNC machines most commonly use G-code, a CNC programming language, to define cutting tool selection, cutting tool feed rate, spindle position, and spindle speed. G-codes are created by processing a cutter source location, or .cls, file defining similar program inputs: cutting tool parameters, position, feed rate, and speed. A key difference between the G-code and .cls file is that the latter contains information on the cutting tool parameters (e.g., diameter, length, etc.); these parameters are central to the CNC program development process.

A typical machining process flow is shown in Figure 1-1. In Orizon's case, CAD files describing the desired product are generated and delivered by the customer. While these CAD files can be delivered in various formats, Orizon often converts CAD files to .stp format due to its universal compatibility. CAM software packages (e.g., NX and CATIA) are then used as an interface to select and visualize cutting tools, paths, and other operation parameters. CAM software generates the .cls file, which is verified in a cutting simulation software (e.g., VERICUT); this verification process ensures there are no major errors with the program (e.g., tool collisions with machinery, etc.). After the CNC program is verified, it is run through a post-processor to develop G-code, which is the language that can be interpreted by the CNC machine. At this point, the CNC machine has all the program data required to fabricate the customer's desired part.

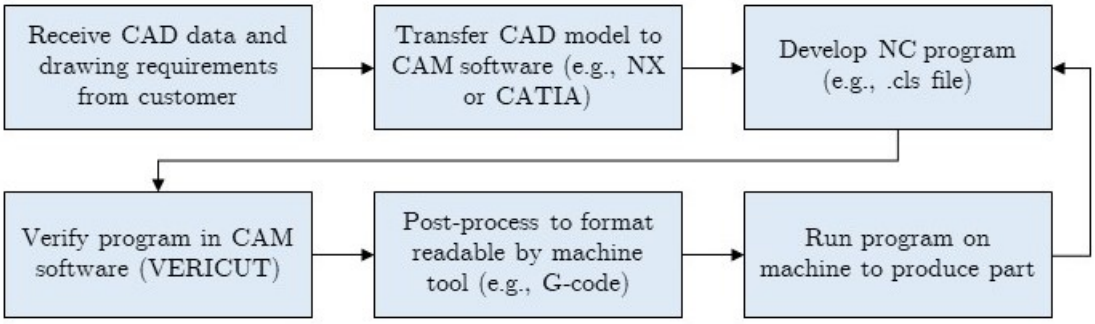

Figure 1-1: A typical process to machine a part specified by a CAD model and other customer requirements includes the use of CAM software to develop and rework the program.

1.3 Onboarding and Programming

1.3.1 Onboarding as a Process

Production optimization and commitment to quality are the main factors differentiating Orizon from competitors in its manufacturing process. Orizon achieves this differentiation with its part onboarding process, which is the workflow required to take a customer requirement (i.e., CAD file and governing drawings) and begin full production on that specific part. A map of Orizon's onboarding process is shown in Figure 1-2.

Engineering activities in the beginning of the onboarding process, including program planning, program creation, and workholding fixture design and fabrication, are perhaps the most notable differentiators for Orizon. Because high volume production work statements are typical, Orizon can optimize EBITDA margins by reducing cycle times and direct labor costs. An investment in NRE to create the optimal production plan and workholding fixture eliminates the need for machine technician labor and reduces quality issues. While some machine shops require a technician to monitor a single machine and adjust machine parameters and workholding in real time, Orizon often requires only a single technician to monitor multiple machines with no real-time adjustments necessary; such adjustments are unnecessary as a result of highly-engineered machine programs and workholding fixturing. Similarly, a commitment to

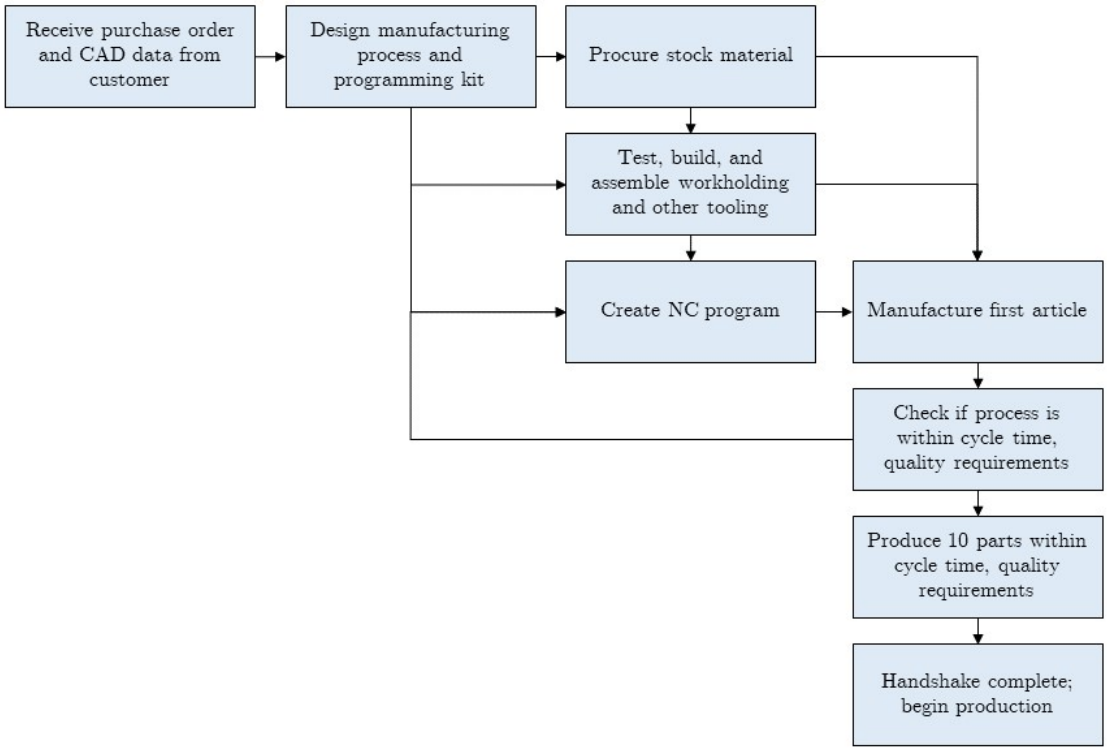

Figure 1-2: Orizon's onboarding process is differentiated with a substantial investment in NRE to reduce cycle time and improve part yield.

designing and fabricating suitable workholding fixtures ensures repeatable processes such that variation in quality output is minimized. This upfront engineering detail is verified in the *handshake* process, which requires 10 parts be fabricated within cycle time and quality (i.e., dimensional tolerance) constraints. With the handshake complete, Orizon proceeds to full production with confidence in its machining process, which is largely defined by the NC program and workholding fixture specific to the part required by the customer.

Such a highly-engineered onboarding process to optimize and verify manufacturing production requires a substantial initial investment in engineering time and labor. This investment varies with the complexity of the part; while simple parts can be put into production in a few weeks or a month, complex parts often require 6 to 12 months. Since complex parts command high prices, Orizon can optimize its EBITDA margin by agreeing to contracts primarily for such parts. By automating the engineering activities in the onboarding process, Orizon can maintain its high standard of optimized, quality production while maximizing its EBITDA margin.

The research herein focuses primarily on methods to automate the NC programming process, which is described in more detail in Section 1.3.2.

1.3.2 Programming as a Process

After Orizon signs a contract to produce a given part, a few activities are kicked off in parallel. Orizon's lead engineers first deliberate to produce a programming kit, which contains a rough outline of how the part will be fabricated on the machine; typically, a programming kit will contain cutting tool choices and major machine operations. Necessary inputs to the programming kit scheme include the raw material definition (material composition, dimensions, and fabrication method) as well as any workholding or manufacturing tooling design. This programming kit scheme is handed off to a programmer, who uses NX or CATIA to develop the program.

Proper selection of cutting tools is critical to the programming process. Consider a simple thru hole drilling operation, which often uses a tool called a twist drill (see Figure 1-3). Some relevant parameters a programmer must consider when selecting

the appropriate twist drill include the following:

- Diameter: the outer diameter of the drill should be equal to the hole diameter (allowing for some tolerance specified by the customer).
- Flute/cutting length: the cutting length should be sufficient to drill the required depth, but be as short as possible to avoid tool chatter.
- Number of flutes: flutes are grooves in a cutting tool that allow for chip formation and removal; the number of flutes (in combination with other parameters) controls the removal rate of material, which should be determined based on the operation, workpiece material, and machine parameters.
- Point angle: the angle of the drill at its tip should vary depending on the material of the work piece and machine parameters.
- Material: the material of the drill should vary depending on the material of the workpiece and machine parameters.

Given the cutting tool definition provided in the programming kit, a programmer will search Orizon's database to find the appropriate cutting tool part number; if Orizon does not already own that tool, the programmer will manually enter tooling data to the appropriate database and work with the relevant procurement personnel to acquire the tool. Note that Orizon currently maintains tooling databases for each site; furthermore, within each site, there is a separate file containing the relevant tooling data for each of NX, CATIA, and VERICUT. As a result, programmers must maintain tooling data and select the cutting tool with the appropriate parameters or risk a costly downstream escape; if, for example, a programmer selects a cutter with longer flute length than intended, the fabricated part may end up having a defect. While program verification steps are in place to mitigate production defects, maintenance of Orizon's cutting tool data via a centralized database would serve to reduce rework in downstream process steps caused by improper tooling selection.

With the appropriate cutting tool data uploaded to the CAM environments, a programmer can develop the machine operations that cut the input raw material

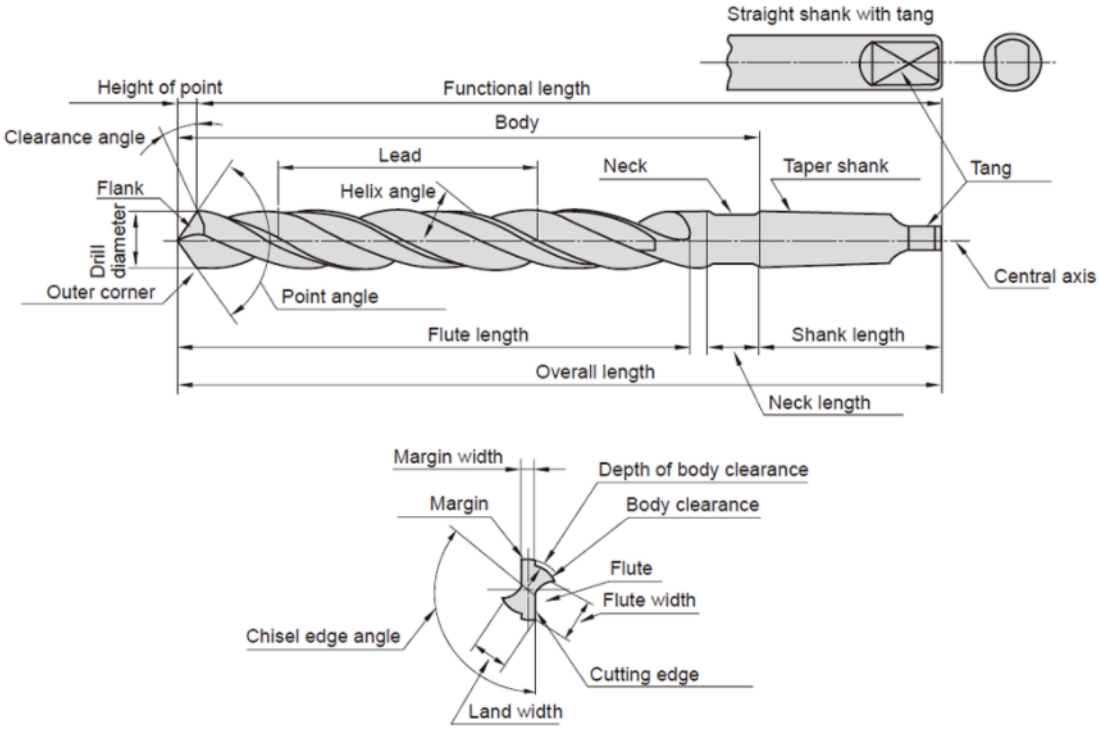

Figure 1-3: A twist drill has several parameters describing its geometry that a programmer must consider when selecting the proper tool [6].

to the desired dimensions. For example, if drilling a simple thru hole in a part, a programmer will select the appropriate tool, the corresponding geometry in the CAD model, and the parameters that govern how the hole is drilled. Some of these parameters for a simple drilling process include the following:

- Feed rate: the velocity at which the cutter is advanced towards the part, often measured in inches per revolution (IPR).
- Spindle speed: the frequency with which the cutter spins, often measured in revolutions per minute (RPM).
- Tool path: the location of the cutting tool in all machine axes over time, which controls how the part is cut.

Note that there are several other parameters a programmer can use to customize a drilling operation. Ultimately, most operations can be pared down to a cutting tool choice, feed rate, spindle speed, and tool path. While NX and CATIA can automatically generate tool paths given certain inputs, developing the operations in the CAM environment can be tedious; a particularly complex part might require 1000 or more operations, each of which takes 30 minutes to program.

Once the program is fully defined, Orizon can simulate the program and verify the program does not result in any part defects or machine issues with VERICUT, a software program providing a digital twin environment for the machining process. If VERICUT simulation reveals any issues with the program, the program can be reworked in NX or CATIA ; in the absence of any issues, a first article will be fabricated in the physical machining environment planned for production.

Notably, Orizon uses a feedback loop to iterate and improve the program after monitoring the first article as it is machined. This process, referred to as development, involves the creation of a *proofing log* as the part is run on the machine. The proofing log captures problematic aspects of the machining program an experienced engineer can sense using visual or audio feedback. For example, the initial tool selection, path, feed rate, and speed might cause the tool to chatter on the surface of the

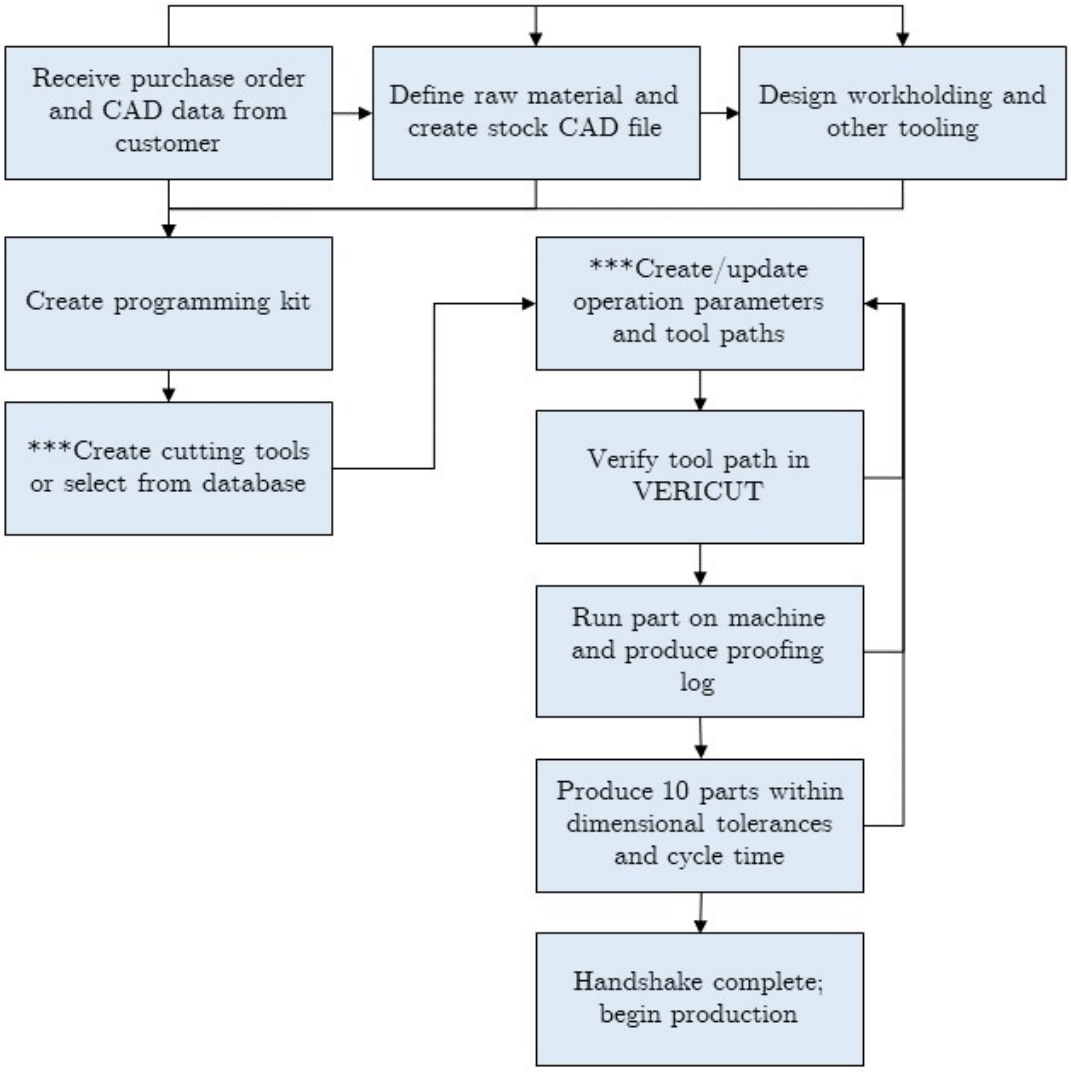

Figure 1-4: Programming is a key part of the onboarding process. Steps indicated by *** are most time consuming and often result in onboarding bottlenecks.

part; this behavior is highly undesirable and is often identified by a particular sound. Since such behavior might risk gouging or otherwise damaging the part, a programmer will edit the relevant program parameters to mitigate tool chatter. Similarly, a programmer can edit the program to ensure the machining cycle time is within the requirement agreed to by the customer and satisfy the customer handshake agreement. The development process and feedback loop are therefore critical to optimizing the machining process and ensuring quality constraints are satisfied. Figure 1-4 outlines the programming process.

1.4 Onboarding Performance

Essential to improving the onboarding process is defining and measuring the key performance indicators (KPIs) such that effects of process changes can be quantified. Consequently, Orizon enterprise resource planning (ERP) data was collected and analyzed to determine the number of hours spent on the programming and overall onboarding processes on a part basis. The accuracy of this data depends on the system each site utilizes to collect ERP data; though Orizon uses SyteLine across all sites, the manner in which SyteLine is used differs between sites. Furthermore, since Orizon focused primarily on achieving revenue growth and recovering from the COVID economic downturn, managers did not focus on ensuring labor data was captured throughout Orizon's tenure.

Nonetheless, onboarding and programming hours collected from SyteLine data were aggregated by part number. A histogram of these data for one manufacturing site is shown in Figure 1-5. At this particular site, a vast majority of programming jobs require less than 500 hours, or roughly 12 weeks. However, there are a few jobs for particularly complex parts requiring thousands of hours. Note that these data represent only a subset of all of Orizon's parts, but are included herein to demonstrate an estimate of the typical programming time and distribution thereof.

Another key consideration to understand Orizon's onboarding performance is the anticipated effect programming automation has on reducing direct labor costs as well as increasing sales. A given part requires a certain direct labor cost related to the number of hours the programmer must spend to create the program; by automating programming, Orizon can reduce the typical cost associated with each part and therefore improve its margin. A more pronounced impact of programming automation is related to the improved sales Orizon can achieve since programmers are a bottlenecked resource. By efficiently producing parts, programmers can focus on new work streams which Orizon would otherwise not be able to produce. Indeed, Orizon often refuses to quote customer requests for work as a result of the bottlenecked programming resources; such a hindrance therefore clearly impacts the company's

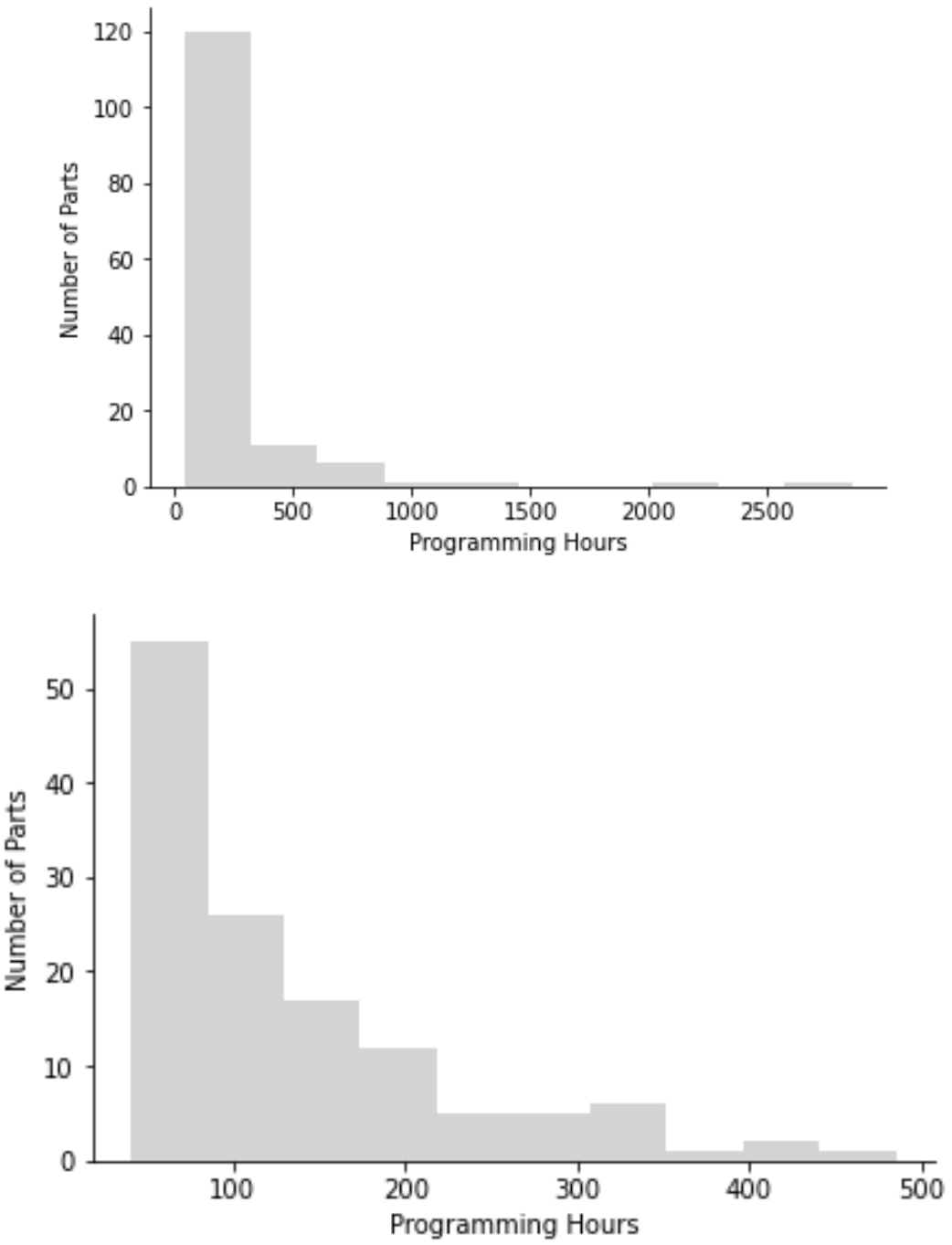

Figure 1-5: Histograms showing the programming hours required for all parts from one location. Plots show all parts (top) and parts requiring programming time less than or equal to 500 hours (bottom).

prospects for revenue growth. Further note that Orizon typically onboards hundreds of parts per year, thereby creating substantial demand for programming jobs and the resulting need to automate programming. The financial impact of these cost savings and increased sales is shown in Table 1.1 and Figure 1-6.

Table 1.1: Table 1: Financial impact of programming automation.

% Automation	Hours saved	Cost savings ($mm)	Sales per programming hour ($)	Increased sales ($mm)
0	0	0	579	0
10	10,000	0.43	643	6.43
20	20,000	0.86	724	14.47
30	30,000	1.29	827	24.81
40	40,000	1.72	965	38.60
50	50,000	2.15	1,158	57.90
60	60,000	2.58	1,447	86.84
70	70,000	3.01	1,930	135.09
80	80,000	3.44	2,895	231.59
90	90,000	3.87	5,790	521.07
99	99,000	4.26	57,896	5,731.74

Note: The results above are calculated assuming a conservative $58mm annual sales for machining sites, 50 programmer heads working 2,000 hours per year, and that programmers remain a bottleneck resource regardless of automation.

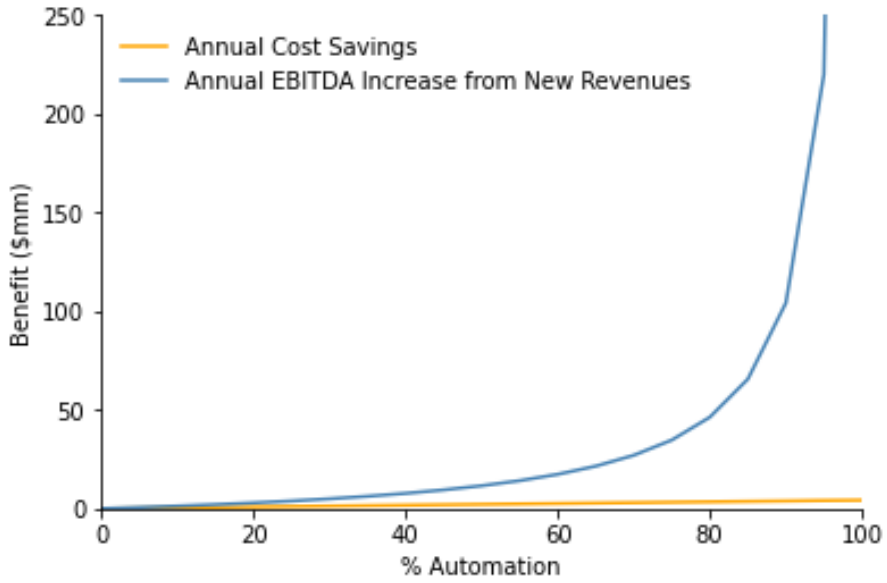

Figure 1-6: Programming automation results in both a reduction in costs and, more impactfully, an increase in EBITDA from improved sales. Note that a margin of 20% is assumed.

At 50% automation, cost savings of more than $2mm and increased sales of nearly $58mm are achievable. At an assumed gross margin of 20%, the combined EBITDA increase of these cost savings and increased sales is nearly $14mm. Furthermore, while cost savings are a linear function of the extent of programming automation, the efficiency of programming hours improves dramatically as programming automation is realized. Therefore, assuming programming remains a bottlenecked resource, sales per programming hour and, as a result, overall sales can increase substantially with programming automation. While it is likely that other process steps (e.g., workholding fixture design and fabrication or machine capacity) would become bottlenecks as the extent of programming automation is improved, subsequent investments can be easily supported with the potential to considerably increase revenues.

Motivated by these operational efficiency and financial gains, Orizon and other machine shops seek to improve their programming processes to reduce time. Though there may be methods to improve the average programming cycle time with investments in labor resources or similar means, this research focuses on technical means to reduce programming time via automation. Prior to presenting novel approaches

of programming automation in subsequent chapters, Section 1.5 discusses current research relevant to the automation task under consideration. Indeed, while 50% programming automation or better may demonstrably improve the financial performance of a company like Orizon, a discussion on whether such a figure is achievable and the method by which it can be achieved follows.

1.5 Review of Current Methods

Integral to the Industry 4.0 concept is the use of digitized data for inclusion in machine learning and broader artificial intelligence algorithms to automate human cognitive processes; in particular, the automation of programming can be considered in two phases: (1) analysis of CAD files for geometrical attributes that inform (2) CAM programs governing cutting tool and operation parameter selection [7]. Current programming automation methods can therefore be broken down into two broad categories aligned with these two phases: feature recognition and operation prediction.

1.5.1 Feature Recognition

A common method of analyzing a part described in a CAD file is to break down the part into its constituent features, such as holes, pockets, and various other types [8]. Note that each of these broad categories of features describes several distinct feature types; for example, simple thru holes, counterbores, and countersinks are all distinct types of holes. A programmer must understand the geometry of these features in order to select the appropriate operation parameters to fabricate the required feature geometry. As a result, feature recognition—the cognitive process of identifying and collecting geometrical data describing the features composing a part—is a critical underpinning to programming automation.

Commercially-available software contains feature recognition technology, but the number of unique features that can be recognized is limited and the technology fails to recognize those with complex geometries [9, 10, 11, 12, 13, 14]. Consequently, while popular CAM software providers such as NX and CATIA make feature recognition

capabilities available to programmers, such capabilities are often not used. This especially holds true for aerospace parts, which typically contain highly complex features unrecognizable by CAM product offerings. For competitive reasons, these CAM software providers do not publicly disclose their feature recognition methods; however, given the following state of academic research in feature recognition, it is unlikely that advanced techniques in academia are used to identify features.

Feature recognition methods utilize one or a combination of multiple of the following approaches: syntactic patterns, graph-based, hint-based, logic rule-based, and artificial neural networks (ANNs); of those five methods, graph-based and ANN techniques are still being explored [15, 16]. Studies of graph-based methods show promise to identify features in complex parts, but only on a simplified subset of parts. Research into the use of ANNs indicate that features can be classified from a supervised dataset of voxelized CAD files and convolutional neural networks (CNNs) with accuracies of at least 96% [17]. Additionally, encoding features into a vector of integers for use in an ANN has been shown to yield greater than 96% feature classification accuracy [18]. Despite their high accuracies, these approaches and others [19, 20, 21, 22] demonstrate success only on a simplified subset of parts and therefore require further development before they can be successfully implemented in a commercially available software designed to recognize machining features in all types of complex parts.

1.5.2 Operation Prediction

Once the features constituting a part and the geometrical attributes defining those features are known, a programmer can determine the operation parameters and corresponding cutting tools to fabricate that part. The determination of those operation and cutting tool parameters by an automated system is referred to herein as operation prediction.

Given the clear benefits discussed in Section 1.4, some CAM software providers have created products that predict operations in addition to feature recognition technology. NX offers a feature-based machining (FBM) package, which allows a user

to create a set of rules to apply operation parameters to recognized or defined features [23]. FBM is the tool used to implement the rule-based approach presented in Chapter 2. In their study of applying FBM at a machine shop, Kruuser et al. report that programming time can be reduced by roughly 20% and programming for about 95% of holes can be fully automated [24]. Some developers also build CAM software add-ons that automatically populate operation parameters based on user-specified criteria in a similar method to that employed by NX FBM [25]. A detailed discussion of the application of these rule-based systems follows in Chapter 2.

Recent academic research demonstrates various methods by which optimization and machine learning can be applied to the machining industry, including operation prediction [26]. Klancnik et al. developed a method employing evolutionary, multi-criterion optimization algorithm NSGA-II to create a program by evaluation in a simulated environment [27]. Dittrich et al. created a routine utilizing material removal simulations and machine learning to automatically compensate for tool deflections, thereby self-optimizing the tool path in a program [28]. Sharmaa, Chawlaa, and Rama developed two machine learning algorithms—one using support vector machines and the other using a restricted boltzmann machine with a deep belief network—to predict G-code for simple drilling operations with up to 97% accuracy [29, 30]. Peddireddy et al. used CNNs and transfer learning to predict whether a part should be machined on a mill and/or on a lathe [31]. While each of these studies reports innovative methods of determining parameters constituting programs, no single study reports a full end-to-end automation that can be readily implemented in practice; moreover, all of these studies focus on simple parts and features relative to those typical of the aerospace industry. Certainly, one or more of these methods may be developed further in future work, resulting in improved practicality to the machining industry.

Chapter 2

Rule-Based System for Program Determination

Rule-based systems are a commonly employed method of AI and a natural methodology to automate decision making for complex cognitive processes [32]. Indeed, a set of questions describing the geometrical attributes of a particular feature can be answered to determine—with some level of accuracy—the appropriate cutting tool and machining operation for that feature. These questions constitute the set of rules defining a rule-based system and are intended to mimic the thought process of a human expert in a fast, highly repeatable computer program. In this chapter, the applicable set of rules to use in such a rule-based system to automate the decision making of a human NC programmer is explored and the Geometrical Rule-based Automation of Programming (GRAP) method is presented.

2.1 Description of GRAP and FBM

In the programming process, a programmer will first analyze all of the features of a particular part to determine the various feature classes inherent to the part as well as the attributes defining the geometry of those features. The programmer will subsequently analyze those feature attributes in order to select the appropriate tool and operation class needed to cut the part on a feature-by-feature basis. This

analysis is composed of learned concepts based on geometrical constraints, physics, and experience; for example, a programmer may choose between two suitable sets of cutting tool and machining operation if one set has been successfully demonstrated in past machining processes.

NX and other CAM software have capabilities to determine both (1) the features and corresponding geometrical attributes for a part and (2) the tool and operation to successfully machine a specific feature based on its geometry (henceforth referred to as feature recognition and tool/operation selection, respectively). These capabilities are typically limited in applicability due to the large number of possible feature-tool-operation combinations as well as the difficulty in recognizing features and their attributes given a raw CAD file. Though limited in applicability with the out-of-the-box configuration, these capabilities may be governed by customized rules and data created by programmers and manufacturing engineers.

NX FBM, introduced in Section 1.5, is governed by the NX Machine Knowledge Editor (MKE), a simple data structure to store user-defined rules and data for programming automation. FBM offers NX users a straightforward interface to compose a set of rules that aid in feature recognition and tool/operation selection. Because roughly half of Orizon's programmers use NX to program parts and other CAM software offerings are not as mature, FBM was selected as the medium to list and apply the set of rules to automate the NC programmer cognitive processes mentioned above. FBM and the specific data owned by Orizon and encoded into the MKE are collectively referred to as GRAP. Note that FBM is limited in its ability to customize feature recognition rules and the focus herein is to implement a set of tool/operation selection rules to replicate a human programmer expert decision process.

In practice, the creation of rules and implementation of FBM will follow the process outlined in Figure 2-1. First, a basic part file containing a simple example of a common feature will be created. This common feature will be recognized via the out-of-the-box NX feature recognition software; otherwise, it will not be a candidate feature for the FBM process. An expert NX programmer will subsequently create the ideal program for that feature (i.e., a tool will be selected and operation parameters

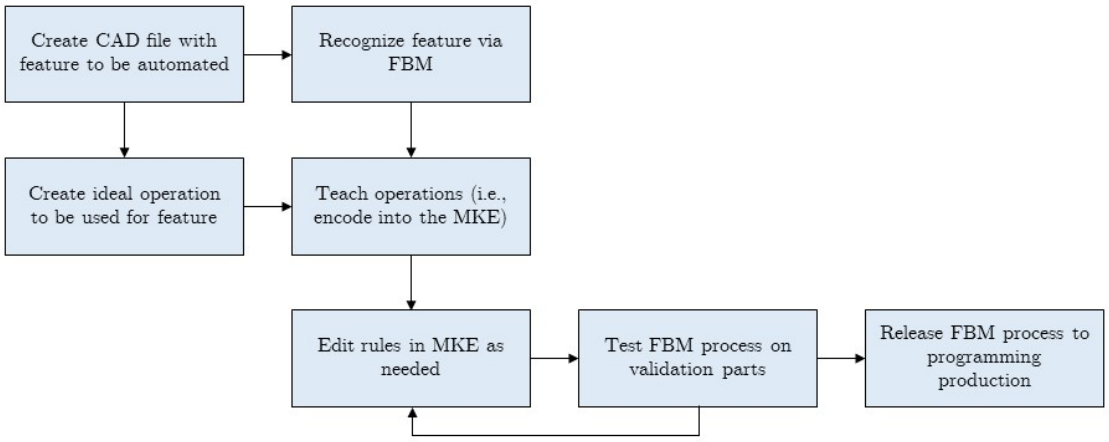

Figure 2-1: The FBM process involves creating a simplified part and corresponding operations in NX, which are then encoded into the MKE and tested prior to application.

specified). This program is then *taught*, or encoded into, the MKE. Because the teaching process is highly specific to the attributes defining only the feature from the simple part file, the MKE rules are edited to ensure general applicability; for example, for a simple drilling operation, the MKE allows users to use IF statements to specify a particular diameter or range of diameters to which the tool/operation selection rules will apply. Once the MKE is updated, an iterative process of testing the rules on new parts and updating the rules in the MKE is undergone to ensure the rules are robust and generally applicable. With an acceptable set of tool/operation selection rules, the FBM process for the particular feature can be used by programmers in their NX environment. Note that for each new part tested or used in production, the out-of-the-box feature recognition rules must be able to identify the feature in said part; if the feature cannot be recognized, then the tool/operation selection rules cannot be applied.

2.2 Case Studies on Pathfinding Parts

Isolated studies of the FBM process were first implemented with pathfinder parts prior to testing the process on a typical part Orizon might machine for its customers.

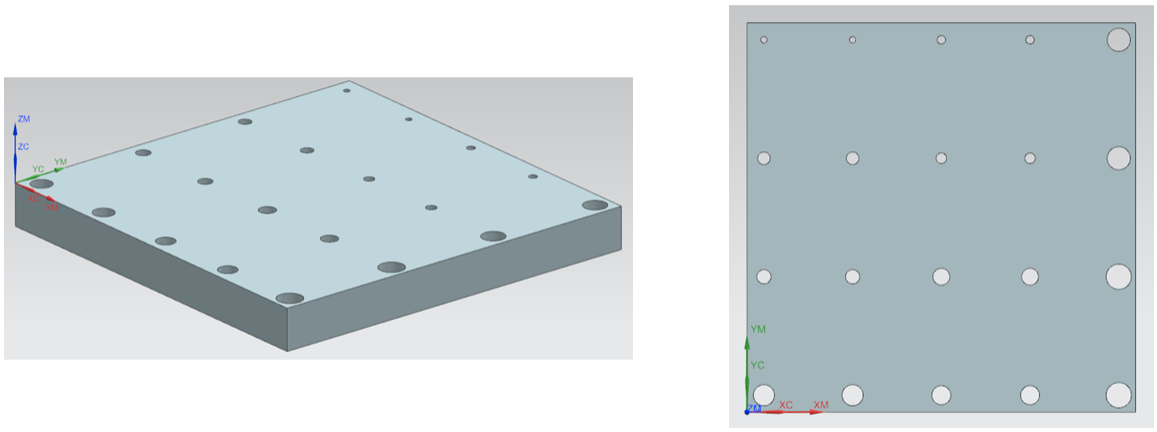

Figure 2-2: CAD model for *FBM1*. Simple thru holes are oriented perpendicular to the top surface of the plate. Hole diameters range from 0.1875 to 0.750.

These pathfinder parts were typically much simpler than a standard Orizon part, which is highly complex due to its end use in aerospace assemblies. Such an approach allowed FBM to be understood in a simple, isolated environment prior to wider-scale implementation.

2.2.1 Pathfinder Part *FBM1*

The first pathfinder part, *FBM1*, is a flat plate with simple thru holes of various diameters ranging from 0.1875 to 0.750 (note: the standard industry convention is that 0.500 will refer to 0.5 inches, and so on). Figure 2-2 shows a picture of the CAD file for *FBM1*.

This initial pathfinder part was chosen because simple thru holes require straightforward drilling operations, so the operation could be easily defined as a starting point to developing GRAP. Additionally, the logic for tool selection requires only two feature geometrical attributes: the diameter of the hole and the depth of the hole.

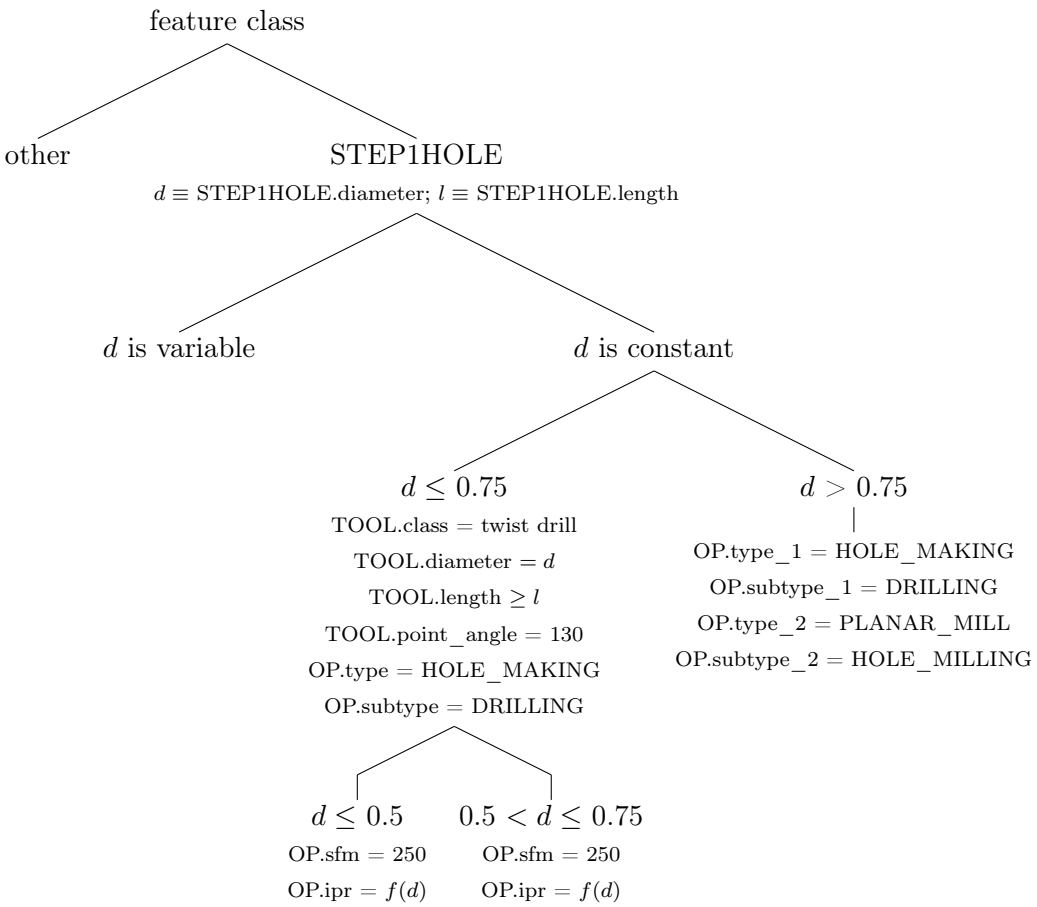

Figure 2-3: Cognitive thought process for choosing a tool and operation parameters for simple thru hole drilling operations. Tool choice and operation parameters are dependent on attributes defined by STEP1HOLE, an NX feature class. Note that aluminum material is assumed and tool material and machine type are ignored.

Using these attributes, a programmer would select the tool and operation according to the rules shown in Figure 2-3. The intent of GRAP is to automatically apply the cognitive process shown in Figure 2-3 such that the programming process—in particular, tool and operation selection—can be rapidly completed.

Given a new part, a programmer will first identify the features inherent to that part. NX STEP1HOLE feature types, which describe various types of holes including simple thru holes, will be identified. A programmer will then investigate the diameter of those features; in particular, if the diameter is constant throughout the length of the hole, the feature will be a simple thru hole (and not a countersink, counterbore, or other hole type). If the feature is a simple thru hole, there are three unique

methods of tool/operation selection that vary with the diameter. If the diameter is greater than 0.75, a drilling operation will be followed by a milling operation to complete the hole. Otherwise, a single operation will be used, but the feed rate varies with diameter: inches per rev (IPR) increases slightly with diameter. However, because the programmer can determine the diameter and length of the STEP1HOLE feature in NX, he can determine the necessary tool and operation parameters to ensure a successful machining job. At this point, the programmer will create an operation in NX from the necessary operation type and subtype (i.e., the template used in NX), select the appropriate tool from the library based on the parameters (i.e., appropriate class, diameter, length, and point angle), and edit the operation parameters as necessary (i.e., appropriate SFM and IPR). Note that, in this simplified example, the workpiece material is assumed to be aluminum and the tool material and machine type are ignored; in general, workpiece material, tool material, and machine type are important inputs governing the machining process.

The rule-based system capturing the cognitive thought process in Figure 2-3 is encoded into the MKE. A screenshot showing rules for simple drilling is shown in Figure 2-4.

FBM utilizes a slightly different logic process than that shown in Figure 2-3. FBM works backward from "OutputFeatures (mwf.)", the desired end state, to "InputFeatures (lwf.)", the feature state prior to any machining operation. FBM logic will match the desired end state to that identified in the feature recognition process; the possible starting state may vary, but is assumed to be blank stock for simplicity. Furthermore, each logic routine—or rule set—references only one "OperationClass (oper.)", which defines an operation type and subtype, and one "Resources (tool.)", which defines a tool class. Should the conditions of the more worked feature (mwf), which is the desired end state, be met, FBM will (1) search for the appropriate tool as defined by the tool attribute rules and tool class and (2) populate a default operation template as defined by the associated operation type and subtype with the operation logic defined within the rule.

In summary, an entire operation can be automatically created from the feature

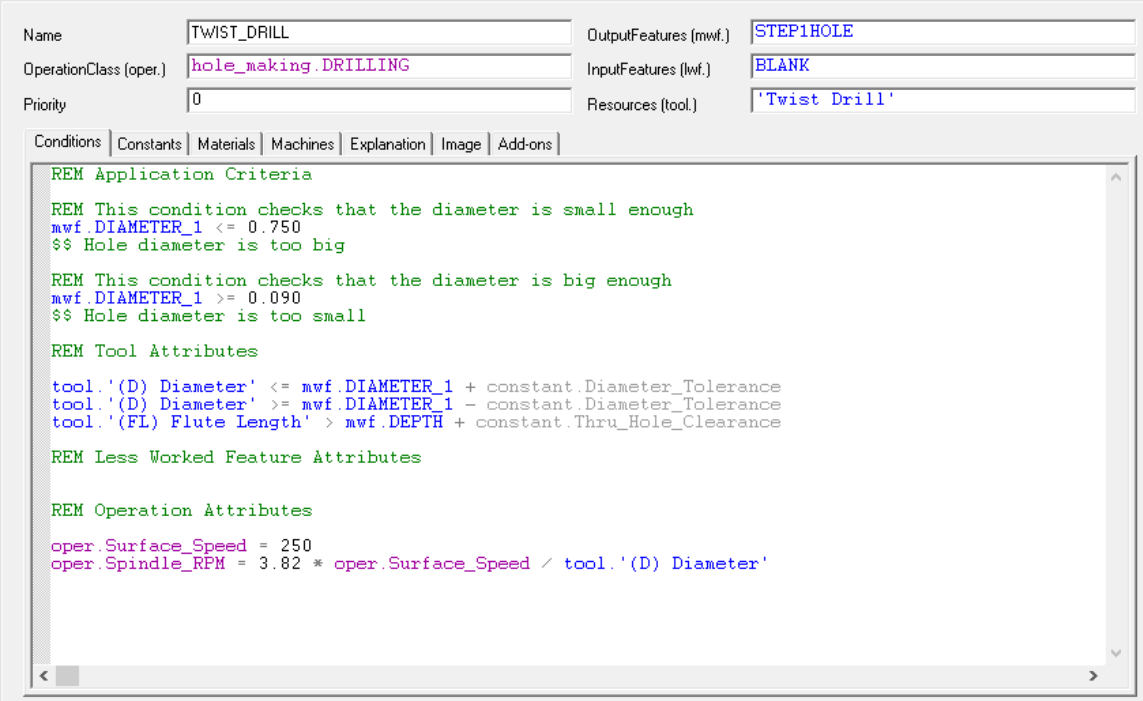

Figure 2-4: Screenshot of the MKE for the simple drilling process. This logic mimics the cognitive thought process outlined in Figure 2-3.

class (e.g., STEP1HOLE) attributes. Indeed, while the manually-created program for drilling all of the holes in the simple plate requires roughly an hour of time, the FBM process produced similar results within one minute. A screenshot of the program FBM automatically created is shown in Figure 2-5.

While the demonstrated time saving for this FBM process is encouraging, there are a few key issues. Notably, the list of selected tools did not exactly match that chosen by the programmer. Additionally, the FBM process failed to identify an appropriate tool for at least one set of holes. The former issue can be attributed to multiple possible tools meeting the criteria of both the programmer and the FBM process; with multiple possible solutions, the programmer selected one at random while FBM produced the shortest tool that met criteria. The latter issue is quite the opposite: no tool met the conditions for the rule set, so the operation was not created; the programmer manually created a digital representation in order to complete the program. Both of these issues highlight the importance of cutting tool database maintenance as described in Section 1.3.2. Were the cutting tool database up to

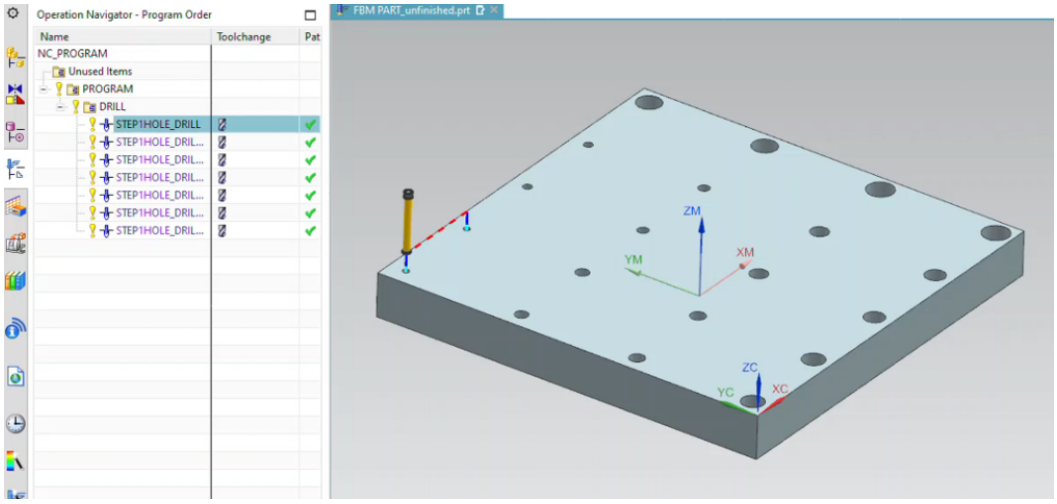

Figure 2-5: Screenshot of the program operations automatically created via FBM using the rules shown in Figure 2-4. The sidebar on the left lists the various drilling operations and tools. The tool and drilling operation for one set of holes is rendered with the CAD model.

date with appropriate tools and devoid of ambiguous tool selections, FBM would have resulted in a better program. Nonetheless, though the FBM process was proven achievable on a plate with simple thru holes, scaling FBM to apply to a variety of complex features typical of Orizon parts proved to be more challenging. These findings are discussed in Section 2.2.2.

2.2.2 Pathfinder Part *FBM2*

Due to the large aerospace composition of Orizon's workload, many of Orizon's parts have pockets. Consequently, given the success in automatically applying drilling operations in pathfinder part *FBM1*, an FBM process was developed on pathfinder part *FBM2*. *FBM2*, shown in Figure 2-6, is representative of a typical Orizon part and has several pockets which serve as ideal candidates to prove out a rule set to automate pocket operations. Additionally, *FBM2* has several simple thru holes which could be used to further test the previously developed FBM drilling process.

Following the process shown in Figure 2-1, simple *teach part* CAD models and programs were created; these parts are shown in Figure 2-7. Each part is a type of pocket—closed-walled or open-walled—that is recognized with out-of-the-box NX

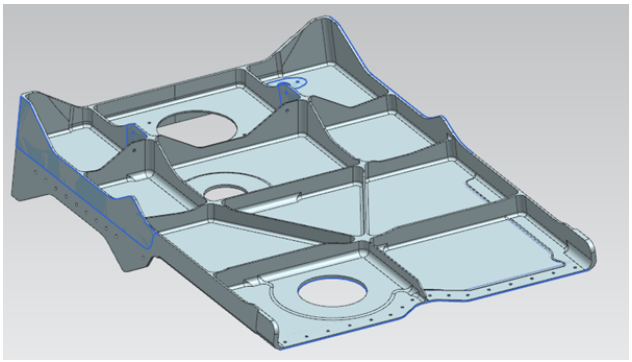

Figure 2-6: *FBM2* part used to demonstrate automation of pocket machining operations in a complex part.

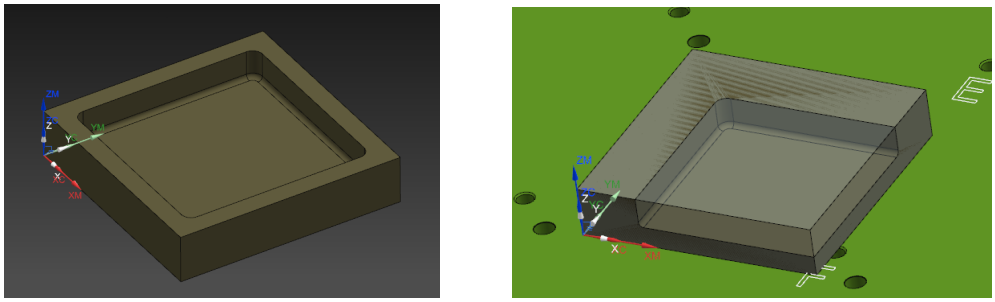

Figure 2-7: Teach part CAD models used to create standard programs for closed pockets (left) and two-sided pockets (right).

feature recognition. Closed-walled and open-walled pockets can be recognized under the NX feature class POCKET_CLOSED and POCKET_OPEN, respectively. The programs for these parts were created using a standard set of operations for a specific machining center used in one of Orizon's three machining facilities. Because the geometries are highly similar, the standard set of operations is equivalent for each pocket type. The list of these operations is shown in Table 2.1.

The operations associated with these parts were then taught into the MKE. Note that the logic for these rules is highly specific relative to the rules for *FBM1* shown in Figure 2-4. A challenge in creating rules for pocket operations is the inability to devise logic—whether simple or complex—to select the appropriate tool for a given operation; the large number of variables to consider (e.g., pocket width, depth, length, wall thickness, etc.) and variability in programmer preferences make devising the required logic infeasible. Consequently, for each operation, a specific tool reference

Table 2.1: Standard operations composing the program for the teach parts shown in Figure 2-7.

Sequence No.	Operation Description	Tool Reference No.
1	Roughing pocket area	M54_A_1B_204_RIP
2	Semi-finish walls	M54_A_1B_204_RIP
3	Semi-finish floor	M54_A_1B_283_SEMI
4	Finish floor	M54_A_1K_355_FLR
5	Semi-finish corner	M54_A_1N_152_MM
6	Finish corner	M54_A_1N_152_MM
7	Finish walls	M54_A_1F_237_WALL

Note: All operations use operation type MILL_CONTOUR and subtype CAVITY_MILL. All tools are stored in the tool library under tool class 5_PARAMETER_MILL.

number matching that shown in Table 2.1 was hard-coded into the rules. While tool selection logic is a preferred method, specifying tool reference numbers directly is a fair approach because a vast majority of pocket operations on a given machining center will use the same tools and operations.

Subsequent application of those rules on *FBM2* had mixed results. The GRAP workflow resulted in successful, automatic creation of operations for *FBM2* pockets; however, the tool paths populated by the FBM process were suboptimal relative to the programmer-specified tool paths. The target pocket operation requires roughly 90 seconds of machine time while the tool path created via FBM would take several minutes. As mentioned in Section 1.2, Orizon leverages the engineering and programming teams to create a high-quality machining process with short cycle times. Therefore an inefficient tool path resulting in a long cycle time is counter to Orizon's machining process approach. Additionally, not all pockets could be found via the NX out-of-the-box feature recognition method, so only two out of eleven pockets had tools and operation parameters automatically created. The resulting tool path for the pocket roughing operations in *FBM2* is shown in Figure 2-8.

2.3 GRAP Implementation Findings

Despite some drawbacks requiring further development, implementation of GRAP via NX FBM was a success in that there is a promising method to automate NC

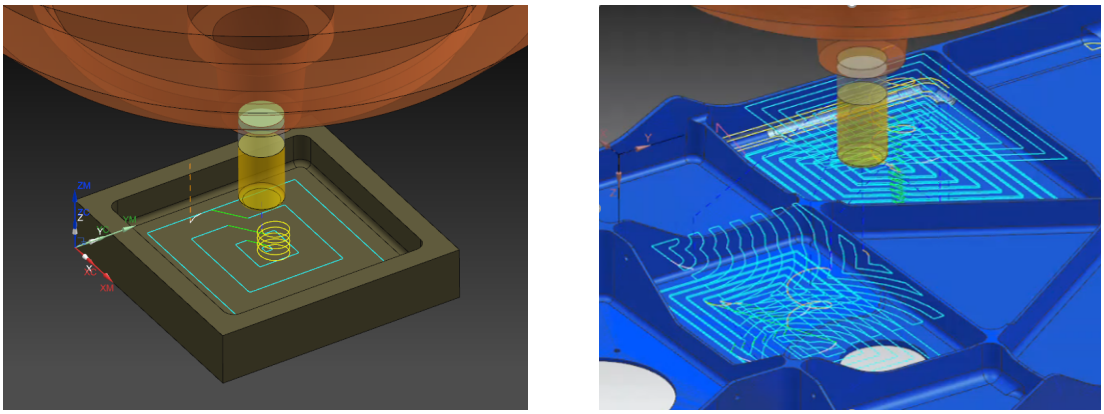

Figure 2-8: Manually programmed, optimal tool path (left) and automatically created tool path for *FBM2* (right) roughing operation. The suboptimal tool path created via FBM defines an inefficient cutting method which will result in a longer cycle time relative to the optimal tool path.

programming using a rule-based AI system. Simple features and related operations such as drilling thru holes are more easily implemented in GRAP than those that are more complex, such as machining pockets in 7 steps. The progress from simple to complex features and operations within a roughly three month time frame suggests that a more in-depth, focused effort with a full-time, expert programmer will yield a robust approach to automation.

Indeed, the personnel working to improve GRAP should include an expert programmer. Rule-based systems are built with the intent to capture expert knowledge; if instead a novice programmer were to create the rules, there might be some poor machining practices captured in the rule system. This could lead to inefficient processes when an NC program created by GRAP is used in production. Not only should expert knowledge be employed in the rule system, but processes should be standardized with a set of common rules. Large manufacturing organizations such as Orizon typically have multiple facilities, each with their unique equipment, protocols, and process methods. As a result, the best practice developed at one facility might not be implemented at another due to acceptance of the status quo and failure to communicate. A rule system captures the best processes at each site in a logic-based format that can be readily shared across sites; such a knowledge base can be used to

document the best, standardized practices of all sites if implemented well.

Similarly, the rule system should be utilized as a means to capture and document expert knowledge. This protects manufacturing organizations from risk in the event that key personnel leave the company. Documented knowledge in a rule system is also an easily-communicated method of training new employees on a particular process—in this case, programming. Nonetheless, documenting such rules is a difficult endeavor. As noted in Section 2.2.2, creating the logic to define tool selection for pocket operations in a robust, scalable method was too difficult, resulting in the designation of a specific tool reference number for each operation. Difficulties arise from a difference in approach between expert programmers and the large number of decision variables to include in rule-based logic.

While determination of a set of robust logic is not necessarily an impossible task, specific designation of rules (i.e., specifying tool references directly) is a more easily implemented solution; since 90% or more of pocket operations will involve a standardized set of tools, specific tool designation will also result in a fairly accurate tool selection process. The few occurrences that require a non-standard tool can simply be edited at the discretion of the programmer. This is also true for the operation in general: results from *FBM2* demonstrate that automatically created operations will often times be suboptimal due to inefficient tool paths or other operation parameters, which can be edited after automatic population in the NX environment. An important facet impacting implementation results from this *semi-automated* approach to NC programming. The current GRAP process rules, which result in an unacceptable program, can be edited or augmented to an acceptable state. Such an approach will still save a programmer a lot of time. The programming process is tedious and repetitive, involving numerous mouse clicks and manual data entry; automatic population of much of the program content reduces this tedious workload such that programmers are auditing and correcting tool paths and operation parameters rather than wholly creating them. This semi-automated approach is a feasible implementation strategy that can standardize and speed up the programming process. Furthermore, this approach allows an AI rule-based system to be implemented to reduce programming

time while a team focuses on developing a more robust set of logic enabling more complete automation.

A few notable drawbacks of the FBM process are inherent to the NX CAM software. Most notably, out-of-the-box feature recognition has poor results; since the FBM process of automatically selecting tools and operation parameters relies on attributes of the desired features, feature recognition is a basic pillar enabling any AI approach. Not only do the attributes defining features need to be used in the rule-based logic, but all features need to be recognized such that tools and operations can be wholly applied. Given the complexity of aerospace parts, feature recognition needs to be sufficiently robust such that all features can be recognized in even seemingly disguised environments.

Additionally, to edit rules in the MKE requires an understanding of a specific coding language used only in NX. This language is not well documented, precluding those who wish to use FBM from efficiently writing, debugging, and editing rules. Furthermore, the data types used by NX are often not sufficient to define a rule in a desired format. The rule in Figure 2-4 specifies that flute length must be greater than the depth, when in fact overall tool length should replace flute length. Similarly, wall thickness of pockets, which are a key programming input, cannot be captured via NX feature recognition. Though all of the GRAP process discussed herein was developed using NX FBM, it is conceivable that a stand-alone rule-based system could be developed; such a system would have robust feature recognition, allow users to access all data attributes, and use a popular, well-documented coding language.

Chapter 3

Machine Learning Method for Program Determination

Similarly to the GRAP rule-based system in Chapter 2, to be consistently integrated with the workflow required by popular CAM software, an ML algorithm should model the human cognitive process of analyzing the geometry of a given part and devising the corresponding program to create that geometry given stock material. An ML approach to programming automation will differ from GRAP in that ML algorithms will result in creation of a model with parameters that were learned from previously created programs. The key motivating factor in investigating such an approach is that manufacturing firms such as Orizon typically maintain hundreds or thousands of programs from parts that have already been fabricated; if an ML approach is successful, such an expansive dataset can be used to train models that predict program parameters rather than devise a set of rules, which has proven to be a difficult—maybe impossible—task.

This chapter presents Deep Learning for Automated Tool Selection (DLATS), an ML algorithm built to analyze geometry of simple holes and select the appropriate cutting tool. A discussion on implementation findings and implications on scaling follows.

3.1 Machine Learning Approach and Results

A novel ML approach was created in concert with NX programming functionality with the intent to automate a programmer's decision on (1) tool class and (2) operation type and subtype. NX allows users to define an operation template, which contains all the relevant parameters to create an operation as long as a geometrical feature and tool choice is specified. Because each operation type and subtype is unique to a template, automated prediction of that type and subtype given input feature geometry would allow a user to automatically create an operation. Furthermore, a prediction of tool class reduces the list of possible tools to simplify the selection process. Therefore, a feasible ML approach would result in a prediction of operation type and subtype as well as tool class given input data on feature geometric attributes.

An initial approach was created to predict operation class and tool class, but due to the difficulty in collecting sufficient data (see Section 3.1.1), this approach was abandoned in favor of an alternative. Because robust data was available mainly for features of NX type STEP1HOLE, only those features were considered in the ML approach. Furthermore, since all STEP1HOLE types had essentially identical operation templates, the approach pivoted to selecting the best tool from among a library of tools given STEP1HOLE feature geometry attributes. Such an approach is analogous to popular methods used by large tech companies such as Amazon. With data describing a user's shopping history, Amazon utilizes a collaborative filtering algorithm to provide recommendations on what a user might be inclined to purchase [33]. By displaying these purchase recommendations in a high visibility area of a user's browser, Amazon can potentially increase the likelihood a customer will purchase a particular item, thereby increasing expected revenues. Similarly, if some ML algorithm is able to recommend a particular tool for a programming job, this recommendation can be automatically populated in the CAM software and thereby reduce the time-consuming process of tool selection central to program creation. A solution to this problem—tool prediction given geometrical feature data—is proposed herein.

An outline of this proposed ML process is shown in Figure 3-1. Feature recognition

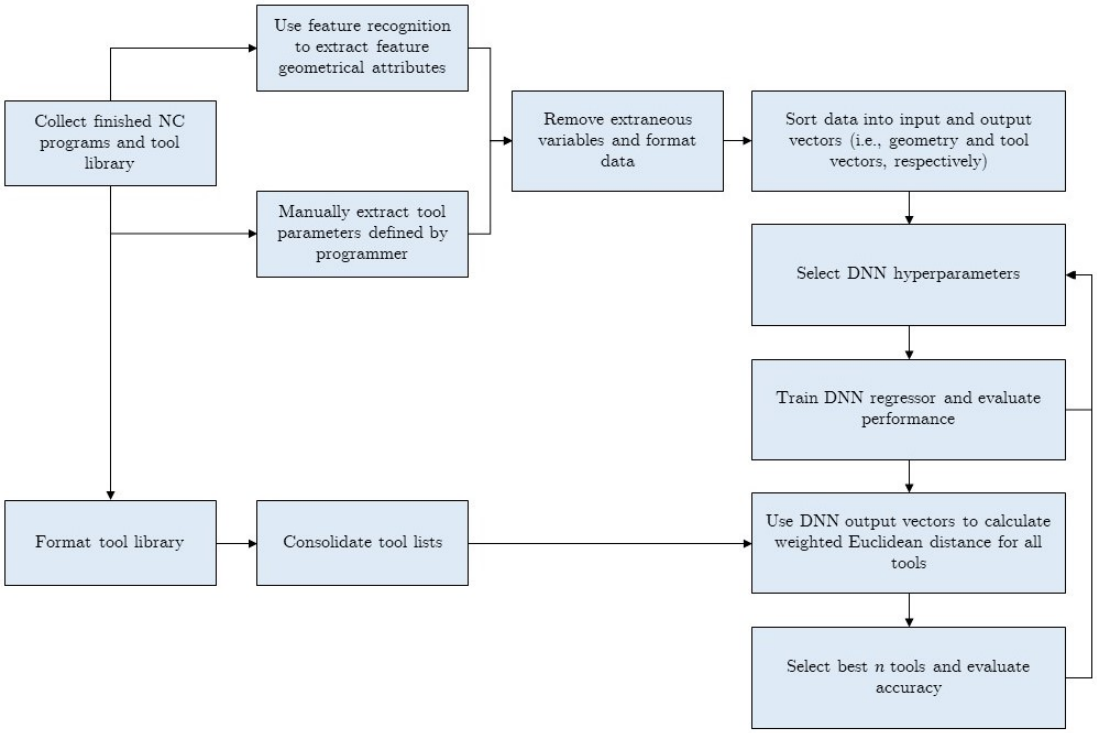

Figure 3-1: Overview of approach enabling the use of ML to predict tools from feature geometry data.

capabilities inherent to NX were used to export feature parameter data describing geometry (e.g., diameter, depth, etc. for a simple thru hole); these data were then put into matrix format, with each row containing a vector of geometrical parameters describing a specific feature. Similarly, corresponding operational data was exported and organized into a matrix format, where each row contains a vector describing two tools—the maximum amount of tools needed for operations considered herein.

A learning algorithm was subsequently trained to predict the vectors describing tools. The parameters comprising the elements of these vectors were then compared to parameters composing the tool library to select the best tool or tools to machine the STEP1HOLE feature based on its geometry. Note that DLATS may output an acceptable tool that is not the exact tool from the sample (indeed, a tool library often has several unique tools that can adequately perform a specific drilling operation); however, a successful prediction is defined as selecting the tool defined in the training sample. Information further detailing this approach appears in the following sections.

3.1.1 Data Collection and Processing

Because ML algorithms are designed to update parameters based on training data, the size, robustness, and quality of the dataset substantially impacts the performance of the algorithm. Consequently, data collection is a key process impacting the success of any ML approach; the quality of the data collected from programs already created by Orizon enables any success a tool prediction algorithm might have.

Orizon's stores of production-quality programs were parsed to collect the necessary feature and tool data. Note that only programs created in NX were used in this dataset because these were accessible files allowing collection of relevant CAM parameters. A Python script was written to parse exported .xml data describing feature geometries while operation and tool data needed to be manually collected in a .csv format. Nevertheless, collecting data from these files proved difficult for a number of reasons.

- Automated collection. NX offers NXOpen, an API allowing users to create custom macros in order to automate CAM operations. An application was created in NXOpen to parse program files for feature, operation, and tool data. However, there was no method to ensure that a given operation and tool sample could be matched up with the proper feature attributes. While some finished programs had easily recognizable feature-operation/tool pairs, most programs did not. Having a method to understand which feature is associated with which operation(s) and tool(s) is a necessary requirement to enable the supervised learning method used herein. Additionally, NXOpen required NX .prt files to be manually opened by a user to allow applications to run. This tedious process was not much faster than manual data collection. As a result, data was collected manually and only 109 examples were collected from 10 different parts.

- Feature recognition. Because collecting the appropriate geometry attribute data requires an NX feature class, feature recognition underlies the data collection process. If a feature is properly recognized, relevant geometry attributes will be populated in the exported .xml file, but may not sufficiently describe the

geometry of interest to the programmer. If a feature is not recognized, a programmer can create a custom feature, but such a process is cumbersome and not standardized.

- Feature variety. 90% of the features in the dataset were NX STEP1HOLE types. Indeed, many parts have several holes drilled into them to aid in assembly following machining; however, due to the inability to collect a large number of samples, this dataset is highly skewed towards STEP1HOLE features. Table 3.1 presents information describing the ML data used.

- Program standardization. Program structure can be defined using one of several unique methods, each of which results in an equivalent program. However, the structure of the program impacts the way data is organized in NX and therefore precludes or enables efficient data collection. For example, a programmer can choose to designate operations and tools under recognized or user-defined features; while this structure would aid the data collection process, most programs are not formatted in such a desirable way. As a result, data collection may be inefficient or even impossible.

These issues preclude the collection of a large, cleanly formatted dataset and perhaps explain in part the lack of widespread success in applying ML algorithms to automatically generate programs. Despite difficulties in parsing NX files, the dataset described in Table 3.1 was collected. Note that, because of the high proportion of STEP1HOLE feature types, the tool selection algorithm described herein was reformatted to consider only STEP1HOLE features.

To prepare the dataset for training, the raw data was filtered to only STEP1HOLE feature class types, operation data was removed, and only certain feature and tool attributes were used. Used attributes were chosen based on physical sense; for example, since tool diameters must match hole diameters for drilling operations, hole diameter is a necessary input for the algorithm. These data were formatted into vectors describing STEP1HOLE geometries (i.e., geometry vector $\boldsymbol{\gamma} \in \mathbb{R}^{25}$) and vectors

Table 3.1: Feature types collected in the ML training data.

Feature Type	No. Features
STEP1HOLE	98
POCKET_RECTANGULAR_STRAIGHT	2
BOSS_ROUND_STRAIGHT	1
STEP2HOLE	2
STEP1POCKET	6
TOTAL	109

Note: Geometry, operation, and tool data were collected for 109 features on 10 unique parts. Of the 109 features (i.e., data samples), 9 samples required 2 tools for the operation; all other samples required only 1 tool.

describing tool parameters (i.e., tool vector $\boldsymbol{\tau} \in \mathbb{R}^9$). Figures 3-3 and 3-2 describe $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ in detail, respectively.

Note that up to two unique tools were considered in the algorithm. This was accomplished by concatenating two tool vectors by sequence order. Since all elements composing a vector describing an existing tool are nonnegative, for an operation that only requires a single tool, the elements describing the second tool were all set to -1.

3.1.2 DNN Regression

A model to calculate an approximate tool vector $\boldsymbol{\tau}^* \in \mathbb{R}^9$ given example pairs $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ was needed in order to predict the correct tool. A natural choice of model to use in this scenario was a deep neural network (DNN), which can regress outputs of a specified length from inputs of an arbitrary length.

The dataset was split into training and test sets, with a test fraction of 33%. A supervised multi-layer perceptron regressor (i.e., DNN) from scikit-learn was fit to the training set and a regression score was subsequently calculated on the test set. An example demonstrating the DNN input and outputs is shown in Figure 3-4. All optimization routines used in this DNN regressor seek to minimize the squared error of the model output relative to supervised training example values.

$$\text{geometry vector} \equiv \boldsymbol{\gamma} \equiv \begin{bmatrix} H_d \\ H_d \\ H_{\theta_b} \\ H_{\theta_{bc}} \\ H_{\theta_{bc_1}} \\ H_{\theta_{bc_2}} \\ H_{\theta_{tc}} \\ H_{\theta_{tc_1}} \\ H_{\theta_{tc_2}} \\ H_l \\ H_{l_b} \\ H_{l_t} \\ H_{l_1} \\ H_{l_{1_b}} \\ H_{l_{1_t}} \\ H_{l_{bc}} \\ H_{l_{bc_1}} \\ H_{l_{bc_2}} \\ H_{l_{tc}} \\ H_{l_{tc_1}} \\ H_{l_{tc_2}} \\ H_d \\ H_d \\ H_{d_b} \\ H_{d_t} \end{bmatrix} \qquad \boldsymbol{\gamma} = \begin{bmatrix} 0.1875 \\ 0.1875 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 1.25 \\ -0.0104 \\ 0.0104 \\ -1 \\ -1 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0.1875 \\ 0.1875 \\ -0.0039 \\ 0.0039 \end{bmatrix}$$

$H_d \in \mathbb{R} \equiv$ hole diameter,
$H_{d_b} \in \mathbb{R} \equiv$ lower diameter,
$H_{d_t} \in \mathbb{R} \equiv$ upper diameter,
$H_l \in \mathbb{R} \equiv$ hole depth,
$H_{l_b} \in \mathbb{R} \equiv$ lower depth,
$H_{l_t} \in \mathbb{R} \equiv$ upper depth,
$H_{l_{bc}} \in \mathbb{R} \equiv$ bottom chamfer depth,
$H_{l_{tc}} \in \mathbb{R} \equiv$ top chamfer depth,
$H_{\theta_b} \in \mathbb{R} \equiv$ bottom hole angle,
$H_{\theta_{bc}} \in \mathbb{R} \equiv$ bottom chamfer angle,
$H_{\theta_{tc}} \in \mathbb{R} \equiv$ top chamfer angle.

Note: $\boldsymbol{\gamma}$ contains some redundant elements and likely some with limited bearing on the final prediction. For example, numerical subscripts indicate additional instances of a similar element. While these elements could likely be removed in future developments, all potentially relevant elements were included for the methods presented herein.

Figure 3-2: Definition of geometry vector $\boldsymbol{\gamma}$ (left) and a geometry vector example (right).

$$\text{tool vector} \equiv \boldsymbol{\tau} \equiv \begin{bmatrix} n \\ T_{1,d} \\ T_{1,l} \\ T_{1,f} \\ T_{1,n} \\ T_{2,d} \\ T_{2,l} \\ T_{2,f} \\ T_{2,n} \end{bmatrix}$$

$$\begin{gathered} n \in \big[1, 2\big] \equiv \text{number of tools}, \\ T_{1,x}, x \in \big[d, l, f, n\big] \equiv \text{parameter } x \text{ of tool } T_1, \\ T_{2,x}, x \in \big[d, l, f, n\big] \equiv \text{parameter } x \text{ of tool } T_2, \\ d \in \mathbb{R} \equiv \text{tool diameter}, \\ l \in \mathbb{R} \equiv \text{tool overall length}, \\ f \in \mathbb{R} \equiv \text{tool flute length}, \\ n \in \mathbb{Z} \equiv \text{tool number of flutes}. \end{gathered}$$

$$\text{One tool: } \boldsymbol{\tau} = \begin{bmatrix} 1 \\ 0.375 \\ 1.75 \\ 0.5 \\ 2 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \text{Two tools: } \boldsymbol{\tau} = \begin{bmatrix} 2 \\ 0.128 \\ 0.965 \\ 0.787 \\ 2 \\ 0.5 \\ 6 \\ 1 \\ 2 \end{bmatrix}$$

Figure 3-3: Definition of tool vector $\boldsymbol{\tau}$ (top) and tool vector examples for the case of one tool for drilling operation (bottom left) and the case of two tools for the drilling operation (bottom right).

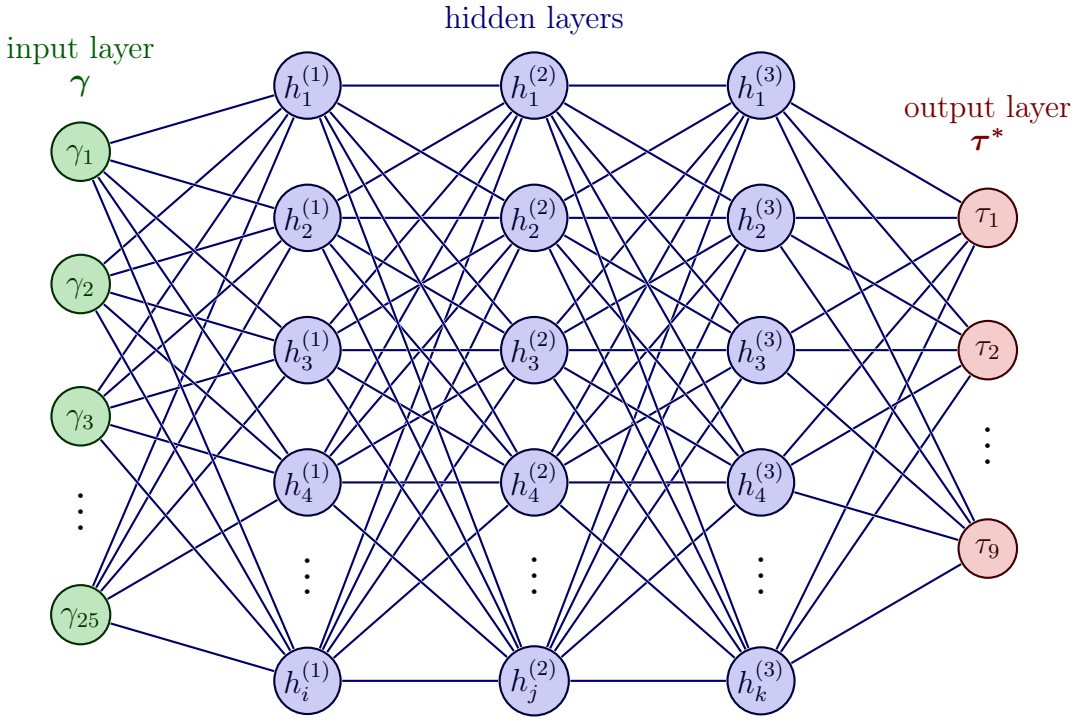

Figure 3-4: Example of DNN using 3 hidden layers of arbitrary sizes i, j, and k. Geometry vector $\boldsymbol{\gamma} \in \mathbb{R}^{25}$ is input to the DNN regressor and approximate tool vector $\boldsymbol{\tau}^* \in \mathbb{R}^9$ is the calculated output.

In order to select the best DNN parameters, such as the size of hidden layers, convergence tolerance, and activation function, a DNN was constructed for each combination of parameters shown in Table 3.2. DNN results were largely insensitive to activation function choice, but a slight improvement in out-of-sample (OOS) regression was evident when using LBFGS in favor of SGD. While a tolerance of 1e−4 yielded improved results relative those achievable by a tolerance of 1e−2, a tolerance of 1e−5 yielded no further improvements.

Hidden layer size had a pronounced effect on the DNN regression results. A subset of these results for which the optimal DNN parameters were used are shown in Table 3.3.

Table 3.2: Parameters used to construct DNNs.

Parameter	Values
Activation function	identity, logistic, tanh, ReLU
Convergence tolerance	1e−2, 1e−4, 1e−5
Solver	SGD, LBFGS
Hidden layer sizes	(25, 25, 25, 25, 25), (100, 100, 100, 100), (100, 100, 100, 100, 100), (1000, 1000, 1000), (1000, 1000, 1000, 1000), (1000, 1000, 1000, 1000, 1000), (25, 100, 100, 25), (1000, 100, 100, 25), (100, 1000, 1000, 100)

Table 3.3: Regression results from various DNN parameters.

Model No.	Hidden Layer Sizes	OOS r^2	IS r^2	Loss	Iterations	Parameters
1	(1000, 1000, 1000, 1000, 1000)	0.843	0.946	0.0095	2895	4,039,009
2	(25, 100, 100, 25)	0.841	0.915	0.0098	1571	16,109
3	(1000, 1000, 1000, 1000)	0.826	0.923	0.0134	1459	3,039,009
4	(1000, 1000, 1000)	0.816	0.922	0.0128	1115	2,037,009
5	(100, 1000, 1000, 100)	0.811	0.933	0.0093	2950	1,205,609
6	(25, 25, 25, 25, 25)	0.804	0.912	0.0092	1567	3,484
7	(100, 100, 100, 100, 100)	0.801	0.921	0.011	712	43,909
8	(1000, 100, 100, 25)	0.76	0.909	0.0115	1420	138,959
9	(100, 100, 100, 100)	0.688	0.953	0.0054	2002	33,809

Note: OOS is out-of-sample and IS is in-sample (i.e., test and training samples, respectively). The input layer had 25 units and the output layer had 9 units.

Unsurprisingly, the best performing model in terms of OOS r^2 is model 1, which contains the most parameters; with hidden layer sizes of (1000, 1000, 1000, 1000, 1000)—equating to a total number of weight and bias parameters of 4,039,009—the achievable OOS r^2 is 0.843. However, model 2, which has 16,109 parameters, over 250 times less than that of model 1, has an OOS r^2 of 0.841, equivalent to only a 0.24% decrease. Furthermore, the difference in the achievable loss between model 1 and 2 is negligible, model 2 converges in half the number of iterations relative to model 1, and model 2 has a smaller difference between the IS and OOS r^2, suggesting that model 2 is not overfit. Model 2 therefore provides the best predictive power relative to its complexity and the time required to build it, as measured by number of parameters and number of iterations to convergence, respectively. Indeed, a more rigorous tuning process may yield a model with slightly better parameters, but the analysis presented herein provides a fair estimate of a robust model.

While relative performance of the models guides the decision of the most appropriate parameters, absolute performance of the model with respect to selecting the appropriate tools determines whether the model is practical in application. Because the models investigated result in regressed elements of a vector of length 9, said vector must be post-processed in order to select the appropriate tool from among a tool library. This tool selection methodology is discussed in Section 3.1.3. Though an OOS r^2 of 0.841 indicates strong predictive power, the ultimate metric guiding absolute performance is tool selection accuracy.

3.1.3 Tool Selection

Because the outputs of the DNN regressor are not necessarily tools that exist in the programmer's tool library, these outputs must be compared with existent tools in order to suggest a useful selection. With an OOS r^2 of 0.841, the DNN regressor in model 2 should output tool vectors that contain elements closely matching those of the training example. However, these elements will not exactly match those of the training example because of the regressed output; consequently, the following method was created to take a tool vector and find the closest match in a library of

Table 3.4: Selection of the best tools from the calculation of the modified Euclidean distances, δ_1 and δ_2, for $\boldsymbol{\tau}^{*-1} = \left[2, 0.122, 0.593, 0.386, 2, 0.526, 5.916, 0.978, 2\right]$. This process results in prediction of A30281.P.H.M.K.N.S for tool T_1 and A30288 for tool T_2 from $\boldsymbol{\tau}^*$.

Tool Ref.	T_d	T_l	T_f	T_n	δ_1	δ_2
A30281.P.H.M.K.N.S	0.125	0.414	0.414	2	0.581	21.603
A30706	0.125	0.414	0.414	2	0.581	21.603
A30254.N	0.098	0.623	0.433	2	0.779	21.594
A30288	0.500	6.000	1.000	2	20.953	0.867
A40288	0.500	6.000	1.000	2	20.953	0.867
A30340.N	0.625	5.000	0.750	2	21.179	4.326

$$\delta_1 = \sqrt{w_1(\tau_2^* - T_d)^2 + w_2(\tau_3^* - T_l)^2 + w_3(\tau_4^* - T_f)^2 + w_4(\tau_5^* - T_n)^2}$$

$$\delta_2 = \sqrt{w_1(\tau_6^* - T_d)^2 + w_2(\tau_7^* - T_l)^2 + w_3(\tau_8^* - T_f)^2 + w_4(\tau_9^* - T_n)^2}$$

where $T_x, x \in \left[d, l, f, n\right] \equiv$ parameter x of tool T,
$w_1 = 1000$,
$w_2 = 10$,
$w_3 = 10$, and
$w_4 = 1$.

tools: a weighted Euclidean distance between the output tool vector and the tool vector of each tool in the library was calculated; the tool with the smallest distance was subsequently chosen as the classified tool. The algorithm defining this method was modified to accept the n tools from the library with the smallest distance to the DNN regressor tool vector output. An example demonstrating this method is shown in Table 3.4 and the corresponding accuracy results are shown in Table 3.5.

Note that the weights mentioned in Table 3.4 were chosen only roughly and purely based on managerial sense: w_1, the weight corresponding to the diameter of the tool, has a high value because tool diameter must match hole diameter within a specified tolerance; w_2 and w_3, corresponding to the tool length and flute length, respectively, have mid-sized values because each length is a relevant parameter, but can vary to a greater extent than the diameter; w_4, corresponding to the number of flutes, has the smallest value, but is still nonzero because many different flute configurations can be used successfully assuming diameter, length, and flute length are sufficient.

Table 3.5: Tool selection accuracy results for select models from Table 3.3.

Model No.	α_{T_1} $(n = 1)$	α_{T_1} $(n = 3)$	α_{T_2} $(n = 1)$	α_{T_2} $(n = 3)$
1	0.226	0.323	0.355	0.355
2	0.161	0.161	0.323	0.323

Note: α_{T_1} $(n = 1)$ refers to the accuracy of correctly selecting the library tool corresponding to the first tool vector with the smallest Euclidean distance. α_{T_2} $(n = 3)$ refers to the accuracy of correctly selecting the library tool corresponding to the second tool vector within the set of 3 tool vectors with smallest Euclidean distances.

The best accuracy achievable for the first tool is 0.323, indicating that there is a roughly 32% chance the selection methodology will have the correct tool within the top three results. Such an accuracy is far too low and the ML process discussed herein needs to be improved before it can be useful in application. However, despite the low accuracy, these results are promising: the probability of randomly selecting the correct tool from a library of 974 unique tools is 1/974, or 0.10%; the ML method discussed herein is therefore about 314 times better than random chance. With modest improvements, tool selection accuracy could feasibly improve dramatically, especially given that the results herein were achieved from a severely limited dataset.

3.2 DLATS Implementation Findings

In seemingly contradictory fashion, results discussed in Section 3.1 suggest that the novel DLATS approach presented herein has reasonable predictive power, yet poor accuracy in selecting the correct tool for a given STEP1HOLE feature. Model 2 achieved an OOS r^2 of 0.841 with only 109 data points; while it is possible that such strong predictive power was achievable only because the DNN model was overfit to the data available, it is certainly plausible that the predictive power will only become stronger as the training dataset is augmented and complete with more sample variability. A dataset with roughly 100,000 samples—an increase of sample size on the order of 1,000 times—would surely be more appropriate for the ML task at hand.

Nonetheless, such strong DNN regression does not guarantee the appropriate tool

is selected. Indeed, the applicability of this ML approach is only useful insofar as high tool selection accuracy is achievable. Were DLATS as documented herein implemented in the programming workflow, the appropriate tool would appear in the top 3 results in only ∼32% of occurrences. While the DLATS suggestion would sometimes be correct and thereby marginally reduce tool selection time, the programmer would most likely need to manually select a tool rather than use one of the options provided by the automated ML system. Furthermore, only tools for STEP1HOLE features would be used, thereby severely limiting the usefulness of such an automated ML system.

Though limited in practical application in its current state, this ML methodology brings to light a completely novel approach to CAM automation, which might be iteratively improved to a state that would provide valuable benefits to the programming workflow. The novel features of this ML approach include the following:

- Tool selection approach. While many people may acknowledge that AI and ML can be increasingly applied in industrial settings, practitioners often have difficulty developing concrete approaches of applying such technology. Existing methods of applying ML in the machining industry do not offer algorithms to be readily used in business applications (see Section 1.5). The research herein establishes a method and algorithm using actual programming data to intelligently and automatically select one or more options from among a library of tools. This approach, though modified from the initial intent, has useful practical application: a time-consuming aspect of any programming job is selecting and loading the appropriate tools into the CAM environment. The selection process can be automated using the ML approach herein, while the loading process can be automated with simple scripts utilizing CAM API macros such as NXOpen for NX.

- Utilization of available data. The ML algorithm presented herein selects the correct tool with far better than random accuracy using only data from previously created programs; all model parameters were calculated using supervised

training pairs as input. This finding unifies ML with the CNC machining industry: it is possible to build an ML algorithm with some degree of predictive accuracy that only requires previously-created programs as a training input. Subsequent improvements may feasibly yield an ML algorithm that utilizes vast amounts of existing program data to predict the correct tools—or other operational parameters—with high accuracy.

- Supervised geometry-tool pair approach. No other documented approach considers feature geometry in vector format as input to a DNN in order to predict the associated tool attributes in vector format in a supervised learning method. Geometry vector $\boldsymbol{\gamma}$ and tool vector $\boldsymbol{\tau}$ were designated with the intent of easily defining an input-output pair to be used with supervised learning. Such a designation is not only a unique approach, but simplifies and standardizes the data to a format that can be easily used in various ML approaches, including, but not limited to, DNN regression. Given the success of this approach with only a limited data set, an improved tool selection accuracy with a more comprehensive dataset may be sufficient to convince leaders of CAM software and processes to reconsider the format in which data is stored; if stored in a format that can be more readily translated to $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ supervised learning pairs or an equivalent, issues with data collection and formatting may be mitigated, ultimately resulting in ML algorithms with high tool accuracies. The ultimate format in which data is represented can critically impact the success of results for complex ML processes and even enable high accuracy models trained with only a few samples [34].

Though this paper establishes the aforementioned useful ML implementation considerations for use in the CNC machining industry, many improvements will be required before any ML algorithm—whether that defined herein or otherwise—will be readily useful. The following necessary improvements are specific to the approach defined herein.

- Data. As for many ML algorithms, data is a key process input and therefore

needs to be abundant and cleanly formatted in order to ensure success. As discussed in Section 3.1.1, relevant data from programs were difficult to collect: programs did not include operational parameters in a format that was easily extractable and paired with feature geometries; a variety of feature types were not available in the programs collected; and no method existed to collect a large enough sample size in an automated fashion. However, with a concerted and targeted effort to parse program files, a large, cleanly formatted dataset can feasibly be collected.

- ML model choice. A DNN regressor model was chosen because it is a straightforward method of supervised learning allowing prediction of output vector $\boldsymbol{\tau}$ from input vector $\boldsymbol{\gamma}$. However, this model is not necessarily the best choice; SVRs, random forests, KNN regression, and other types of models may yield better predictive results. Though a discussion of various model types is not included herein, because of the simple input-output pair data format, the success of these models can be easily investigated and compared to the results achieved using the DNN regressor model.

- Tool selection algorithm. The weighted Euclidean distance approach may not be a sufficiently robust method of tool selection; in addition, the specific weights used in the calculation methodology (see Table 3.4) may not be the optimal weights yielding the best OOS tool selection accuracy. Perhaps an improvement in the tool selection methodology would better utilize the high performance of the DNN regressor model and subsequently improve the tool selection accuracy. A detailed study of this method is not included herein.

- Elements of $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$. Section 3.1.1 mentions that the specific elements used in $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ were selected based on physical sense—meaning that elements describe physical quantities that a programmer might be expected to use when creating a program. For example, a programmer cannot select the appropriate tool for a simple drilling operation without knowledge of the diameter and length of the hole to be drilled. Therefore, at a minimum, $\boldsymbol{\gamma}$ is required to have two elements:

one describing hole diameter and the other describing hole length. However, NX STEP1HOLE feature types have a rich list of attributes beyond hole diameter and length. A more rigorous investigation into the appropriate constituent elements of $\boldsymbol{\gamma}$ may indicate that more descriptive, physically-grounded attributes are required in order to improve tool prediction accuracies. Similarly, $\boldsymbol{\tau}$ may be more easily predicted if it contained more elements than those used herein. This research is limited to applications via NX and the attributes defined for STEP1HOLES and other feature types; with a more comprehensive set of feature attributes, $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ vectors may be more descriptive and yield improved predictive results.

- Feature complexity. The ML method developed herein focuses solely on STEP1HOLE feature types. Machine shops will typically be exposed to a more comprehensive set of feature types; consequently, the usefulness of this method is limited. Before becoming useful in practice, this method will need to be expanded to include a few repetitive features, such as pockets, wall tops, mold lines, and undercuts in addition to holes.

Important to the evaluation of this ML approach is the tool accuracy calculation. Tool accuracy judges whether the tool prediction matches that used in the program; however, it is possible that the ML algorithm output might suggest a tool that sufficiently accomplishes the programming job—or suggests a better tool than that used by the programmer. Tool accuracy might be improved by considering not whether the exact tool reference number is predicted, but whether the parameters of the tool are sufficient to complete the drilling process as intended. Such a determination, however, is likely too complex: a set of rules similar to the rule-based system discussed in Section 2 would need to be created in order to evaluate whether the parameters of a specific tool could be sufficient. Because it is feasible that, with an improved ML algorithm, predicted tools will be sufficient to complete a programming job, but not match those create by the programmer, a system will need to be devised in order to verify whether the elements of $\boldsymbol{\tau}$ or its equivalent are sufficient.

Indeed, this ML approach can only create value if implemented at scale within Orizon. Implementation at scale requires moving from the *bench-top* algorithm to a system that is used nearly ubiquitously by programmers. Tool accuracy will need to be improved prior to full-scale implementation; as a result, continued research into the approach with the intent to correct the improvements mentioned above is a required next step. The software in which this algorithm is embedded also largely impacts the success of implementing ML—whether the algorithm discussed herein or otherwise—at scale within Orizon. Such a software system must (1) provide a clean and abundant source of data to feed the algorithm; (2) deliver the algorithm to the users (i.e., programmers) in an interface that is easy to use (otherwise, programmers will less likely adopt this new ML method); and (3) allow modularity such that improvements to the algorithm can be made in quick fashion. A commitment to implementing ML at scale therefore requires a commensurate investment in this software infrastructure, which can be difficult if personnel experienced in such systems are not available.

An equally important consideration to scaling is algorithmic robustness, meaning the success with which it can be applied in different environments. For example, Orizon is composed of three major machine shops, each of which has multiple machining centers. These machining centers have different operating parameters and cutting tools; as a result, the specific operation and tool used to cut a pocket in one machining center may vary from that used to cut the same feature in a different machining center. While some of these process deviations are required due to the hardware at hand, much of the deviation results from the nature of the company; Orizon was built partially through acquisition of different machine shops, each with its unique set of people, processes, and tools. Orizon management has undergone concerted efforts to reduce such deviations and consequently simplify the programming process. Insertion of an advanced technology such as ML into the process may complicate matters, but might also help standardize processes: at any given time, the ML model will output the same cutting tool suggestion for the same input feature geometry attributes. Where processes must differ between machining centers, a separate algorithm, specific to a machining center, can be easily trained. Therefore, while it is possible to build a

robust set of ML algorithms that standardize programming processes, doing so while simultaneously building a supporting software infrastructure might prove to be an arduous task.

Therein lies the crux of the decision faced by management: though an investment in a high-tech ML approach has the potential to fundamentally and dramatically improve the programming process to reduce overhead costs and product velocity to market, such a technology may fail and requires support by a software infrastructure beyond the firm's core capability to machine and assemble complex aerostructure assemblies. To a limited extent, the research herein increases the probability of success in implementing ML, which makes the investment more favorable; nonetheless, such a high-stakes managerial decision necessitates a more detailed analysis to determine whether the potential financial gains and strategic positioning are in favor of the company.

Chapter 4

Comparison of Approaches to Programming Automation

Two general approaches to programming automation are presented herein. GRAP utilizes a rule-based system in order to select the appropriate tool and operation parameters given a CAD file with defined feature geometries. DLATS involves an ML algorithm that selects a drilling cutter from among a library of tools based on geometrical attributes of the hole to be drilled. This chapter compares each of these approaches.

Important to note for this comparison is the stage of development for each approach. GRAP and DLATS are both in a bench-top phase of development, meaning that further research is required to improve their governing rules and algorithms until acceptable results are achieved. Therefore, it is possible that improvements may result in one approach becoming more favorable than the other at a later date; nonetheless, this evaluation is written under the current understanding of each approach.

Further note that there are different inputs and outputs specific to each approach. GRAP is built entirely in the NX environment and consequently has the ability to parse entire CAD models for relevant geometrical features (to the extent those features can be recognized). As an output, GRAP will produce a complete (though not necessarily correct) program, including cutting tool designations and operation parameters. In contrast, DLATS requires exported data describing geometrical features

as an input and provides n cutting tool recommendations as an output. GRAP therefore provides a more complete automation process that is more readily implemented because it already exists in the NX environment. However, due to the issues stemming from creating a rich set of rules, GRAP is not easily customizable relative to the software infrastructure a manufacturer would need to build to integrate DLATS into the programming workflow. These inputs and outputs, a key consideration relevant to the following comparison, are the most significant differentiators of GRAP and DLATS.

While it is feasible that the results produced by GRAP or DLATS can be used in some semi-automated fashion to reduce programming time, the usefulness of those results improves with accuracy. Consequently, the accuracies of GRAP and DLATS are an important point of comparison. The accuracy achievable by DLATS is well understood: at best, tool selection accuracy is 32% for tool T_1 and 36% for tool T_2. This means that, more often than not, a programmer using DLATS will have to manually choose the appropriate tool, unless accuracy is improved. GRAP accuracy is more nuanced: there is some accuracy of feature recognition, measuring the success with which a given feature type is recognized, and operation prediction, measuring whether the correct operation was automatically created in the program. Feature recognition accuracy in GRAP varies significantly by part and feature; 100% of STEP1HOLE features were recognized in *FBM1* whereas 2/11, or 18%, of pockets in *FBM2* were recognized. Indeed, the accuracy of pocket feature recognition could be improved by continuously teaching new feature geometries into the rule base, but feature recognition should be robust in implementation. As a result, feature recognition accuracy is low. Operation accuracy is also unfavorable; none of the automatically-created tool paths in *FBM2* was identical to those created by the programmer. Though it is possible to edit the programs created by GRAP, the accuracies of feature recognition and program creation need to be improved. At the current phase of development, both GRAP and DLATS require programmer intervention in order to finalize the program. However, GRAP only needs minor modifications to ensure the correct operational parameters are used in program creation. Therefore, with the usefulness

of each approach limited by the capabilities of feature recognition, the operation application accuracy of GRAP will likely outpace the cutter tool selection accuracy of DLATS in later phases of development.

Because the purpose of automation in this context is to reduce the time a programmer spends creating a useful program, the speed at which either GRAP or DLATS produces an output is another relevant point of comparison. Without the supporting software infrastructure that enables DLATS, the speed with which this approach provides cutting tool recommendations cannot be determined. However, it is feasible to assume that the recommendation will be near-instant: the parameters of the DNN regressor are all pre-trained and the algorithm needs only to calculate output vector $\boldsymbol{\tau}^*$ and the corresponding n tool recommendations from a library of only roughly 1000 tools. Depending on the supporting infrastructure that enables programmer use, these simple calculations may be augmented with additional software routines that reduce overall speed. Despite its integration to the NX environment, analysis of the entire CAD model, and output of a complete program, GRAP takes less than one minute to recognize features and produce corresponding programs. Though this is slightly slower than the time achievable by DLATS, each process can be considered near-instant relative to the timescale of a typical programming job, which typically requires 40 to 1000 hours. If each method matures to a level enabling accurate creation of complete programs, reducing speed to sub-minute timescales may become a concern warranting further investigation.

The larger system of which GRAP and DLATS must be a part not only affects the speed of program creation, but the ease of implementing either approach at scale. As previously mentioned, a supporting software infrastructure will need to be built such that DLATS can be integrated into the programming workflow. Such an infrastructure will need to integrate with popular CAM software such as NX or CATIA. While possible with available software development kits, this requires a concerted effort of software engineers and application developers, both of which are not typical employees for a manufacturing firm like Orizon. GRAP, because it exists entirely within NX, is more easily implemented, but still has some relevant hurdles. The

rules defining feature recognition and operation application, which are written in a custom NX programming language, need to be controlled and maintained by an expert NX user. NX users are typically more familiar with the programming interface than the computer programming language required to customize the rules. NX does provide a user interface that allows rules to be taught and controlled in the environment familiar to the NX programmer, but customizing rules—which is required to build a robust system—requires using the custom language, which is poorly documented. Nonetheless, GRAP is more readily implementable than DLATS because it is currently available and supported within NX.

Another important implementation consideration is scalability: each method has been developed only on a bench-top scale, an environment substantially different from the target production scale. Production scale is defined as full implementation in the programming workflow at each manufacturing facility for a given manufacturing firm. Notably, each manufacturing firm may maintain a unique set of tools and operational parameters to produce an equivalent feature; consequently, successful achievement of production scale will ensure the ability to maintain heritage processes, but also assist in unifying the manufacturing firm to a common set of practices such that programming is simplified. A tool selection and operation application scheme can be defined via rules in GRAP and consequently held as the standard to machine a particular feature; in addition, these rules can be copied and edited should any variations be needed (i.e., heritage processes differ between manufacturing facilities). A designated expert at each site can maintain the rules constituting GRAP in order to ensure feature recognition and operation application processes are best employed. Because DLATS is a machine-learned model, there are no rules that need to be updated. Rather, the model will be trained with the specified input data and will provide a consistent, standard cutting tool selection for an input feature geometry. While this serves to unify the processes among various manufacturing facilities in the same firm, separate ML models can be easily maintained such that each facility has its unique process, if desired. Therefore it is feasible that both GRAP and DLATS can be implemented in a production scale.

In contrast to a rule-based approach, an ML approach enables the analysis of previously created program data to develop a model that predicts program parameters given input feature geometry. Because ML models evolve with the dataset used to train their parameters, a model that predicts program parameters can be trained with a data warehouse of previously created programs; such a model can also be fine-tuned by only using optimal program processes in the training set. This enables a company such as Orizon, which has developed several hundred programs over its tenure, to utilize the work performed by programmers in a more efficient manner: programs are used not only to fabricate parts, but to train a predictive model to improve programming efficiency. A rule-based system such as GRAP requires continuous maintenance to ensure the constituent rules appropriately capture training methodologies and can only indirectly and narrowly utilize the previously created programs through the experts that create the rules.

An additional advantage of ML is that complex mathematical formulations can be replicated using learning techniques such as DNNs. A rule-based system requires an expert to determine a comprehensive set of simple formulas that, when considered in full, result in a highly complex decision process and mathematical formulation. Given the appropriate training data, ML algorithms can learn such a complex decision process, thereby eliminating the need to create an entire set of rules, which can be an arduous and sometimes impossible task. Indeed, in the development of GRAP, rules could not be created to automate a seemingly simple process such as pocket machining because programmers could not agree on the best machining method and there were too many intricacies and nuances in the programmer's cognitive process.

There are certain straight-forward processes, however, for which a simple rule constraint can perform more favorably than an ML prediction. Consider the selection of a cutting tool in the DLATS algorithm; the diameter of the tool should match that of the hole. A rule can quite easily be devised to ensure this particular constraint is satisfied (see Figure 2-3). The same is not true for DLATS: instead of directly specifying a constraint for the tool diameter, the algorithm trains the DNN regressor parameters in order to create a prediction $\boldsymbol{\tau}^*$ with elements $T_{1,d}$ and $T_{2,d}$ that closely

Table 4.1: Errors in tool diameter prediction from DLATS test samples.

Tool (T_1 or T_2)	d (actual)	d^* (predicted)	$\lvert d - d^* \rvert$
T_1	0.1280	0.1216	0.0064
T_1	0.2010	0.2105	0.0095
T_1	0.7500	0.5369	0.2131
T_2	0.5000	0.5258	0.0258
T_2	0.3750	0.3041	0.0709

Note: Data shown from test sample only. $|d - d^*|$ had a mean of 0.0360 and standard deviation of 0.0378 for 31 total T_1 test samples. For 2 T_2 samples, the mean and standard deviation were 0.0483 and 0.0225, respectively.

approximate the elements of $\boldsymbol{\gamma}$ describing hole diameter. The DNN regressor "learns" that tool diameter should match hole diameter because the aforementioned elements of $\boldsymbol{\tau}^*$ should be close in value to the associated elements of $\boldsymbol{\gamma}$, as governed by minimizing loss between $\boldsymbol{\tau}^*$ and $\boldsymbol{\tau}$. As shown in Table 4.1, the average error in d for T_1 was 0.0360, meaning that the predicted tool diameter will differ from the actual tool diameter by 0.036 inches on average. Tolerances on simple drilling processes should typically be within 0.002 inches.

Should variations of DLATS or other ML algorithms demonstrate success in correctly learning such rule constraints (i.e., due to better data and predictive models), it is conceivable that those algorithms could be improved to learn more complex constraints that cannot be easily created in a rule-based system such as GRAP. This is a major advantage of pursuing an ML approach such as DLATS in favor of a rule-based approach such as GRAP: given access to large datasets, computational speed, and the appropriate algorithms, complex programming rules ought to be learned, not specified. Otherwise, complexity will preclude the number of useful rules to be used in programming automation and, consequently, the jump from automating simple programs such as hole drilling operations to more complex programs such as pocket machining will not be made.

A rule-based system such as GRAP can be readily implemented and produces an output which is integrated into the NX environment, thus demonstrating some ap-

plicability to the ultimate goal of program automation in its current developmental state. However, the benefits of ML—that complex processes can be learned from stores of program data and applied on new geometries instantly—outweigh the drawbacks that tilt favor towards a rule-based approach. Though both approaches can be carried forward, this analysis suggests that an ML approach will yield a more intelligent, adaptable, and robust means of program automation in years to come. Below are some concerns of an ML approach and corresponding mitigating factors as evident in the analysis of DLATS.

- Limited capability and low accuracy. DLATS only outputs a suggested tool for a simple drilling operation and therefore has limited utility in a production environment. With further development, this capability can be extended; a team of data scientists internal to a manufacturing firm can continue to develop sound algorithmic approaches to augment ongoing academic research.

- Data availability and software infrastructure. Lack of clean data is a well documented flaw to DLATS, yet a relatively minimal concern to GRAP. However, a concerted effort to create a robust data architecture will mitigate this concern as manufacturing firms bolster their IT departments with the skillsets required to realize an Industry 4.0 vision. These IT personnel will not only create the necessary data architecture, but the supporting software applications that deliver the result of an ML model such as DLATS to the programmer in the appropriate environment.

- Explainability. Many ML algorithms, especially DNNs, are criticized as *black boxes* offering minimal insight into how results are obtained. Rule-based systems, on the other hand, have results that can explained in the context of a simple decision tree. In recent years, there has been a dramatic increase in the number of studies focused on explainable artificial intelligence, including DNNs [35]. While the research herein does not include an investigation into how the DLATS algorithm might be explained, the recent focus in explainability serves

as a potential mitigating factor to this problem for future ML developments related to CNC machining.

- Easily fooled. Some ML algorithms can be easily fooled, meaning slight perturbations in an ML input can lead to dramatically different outputs, often of low accuracy [36, 37]. Because this effect is dependent on the data representation and algorithm employed, a detailed study of future ML developments related to CNC machining can be conducted to determine the extent to which this issue might affect applications.

Chapter 5

Conclusions and Future Work

Orizon, and presumably most other manufacturing organizations, have long recognized the potential to realize efficiency gains with programming automation via reduced overhead expenses and increased revenue per programming hour. However, the numerous challenges presented in Chapters 2, 3, and 4 demonstrate the difficulty of implementing a reliable AI solution to the problem of automating programming. The difficulties evident in this research can be summarized into a few major hurdles: processes and CAM software packages used by the machining industry preclude efficient collection of relevant data and integration of AI applications; the skillsets inherent to machine shops largely differ from those required to build a useful software-based AI application; and the CAD geometries and associated machining processes are highly complex, making full automation a major challenge.

A useful comparison to the problem of automating programming is autonomous vehicles, particularly the pursuit to develop, produce, and sell self-driving cars to be used by the typical American consumer. Autonomous vehicles have been in development for decades, but have only just recently made breakthroughs due to availability of data, improvements in computing power, and advances in AI algorithms enabled by increased investment in the space; despite these improvements, no fully autonomous car is currently available for purchase by the common consumer and use on regular roads [38, 39]. This inability to develop such an autonomous vehicle is arguably the result of the complexity of the problem at hand: an AI algorithm must have the

ability to predict the correct response (e.g., steer, brake, accelerate, etc.) to an infinite combination of environmental inputs (e.g., stop signs, turn signals, pedestrians, etc.) with near-perfect accuracy. If not, the technology for self-driving cars would likely not be adopted due to safety issues. Similarly, the problem of programming automation using AI is highly complex. There are infinitely many combinations of CAD geometries in addition to an infinite number of methods to machine a given geometry. Indeed, the problem of programming automation is arguably more complex than that of autonomous vehicles since there are a limited number of actions a car can take in response to its environment. Nonetheless, each process is highly complex and the inability of a focused development to produce reliable self-driving cars to date perhaps explains the similar lack of success in programming automation.

The available labor skillsets in the automobile and machining industries also reveal an interesting point of comparison. The advent of AI systems in automobiles requires that the industry's skillsets shift to include software engineers, data scientists and engineers, and AI experts, which will presumably be a significant aid in the ongoing progress of autonomous vehicles to a fully-deployed state [40, 39]. Such a shift in skillsets and targeted investment in AI technology has not yet gained the same momentum in the machining industry; however, if the automobile industry is any indication, including the appropriate skillsets in AI projects in the machining industry can lead to similar developmental strides enabled by efforts to improve data collection and create a software infrastructure allowing integration of AI algorithms into the programming workflow.

Additionally, the approach to autonomous vehicles varies greatly between automotive companies such as Tesla and GM. Tesla's Autopilot utilizes cameras and radar, whereas GM's Super Cruise uses LIDAR for driver assistance; each approach is *semi-automated* as the automobile manufacturers seek to develop a fully-autonomous system [41]. The semi-automated approach of these companies can be useful inspiration for those seeking to automate programming: rather than slow progress towards a fully-automated solution, perhaps incremental developments to automate some programming processes utilizing unique approaches, then improving from those processes

further to increase the level of automation, may prove to be a more useful means to obtain a compelling solution.

Regardless of the approach, the research herein presents two methods of programming automation that may dramatically improve machine industry operations with a concerted and continued development. The following sections discuss feature recognition, a fundamental building block of programming automation, and other AI/ML methods to be considered moving forward.

5.1 Feature Recognition

The analogy comparing programming automation to autonomous vehicles can be extended further to demonstrate the importance of feature recognition. Autonomous vehicles require robust computer vision in order to categorize road obstacles (e.g., stop signs, speed bumps, etc.), which subsequently affect the appropriate response of the vehicle [38]. Similarly, a programming automation algorithm must first analyze a CAD file for geometric features and collect data on their attributes. This is a crucial step in the process: an input describing the geometry of a feature, such as $\boldsymbol{\gamma}$ defined in Chapter 3, enables a supervised learning approach which mirrors the cognitive process of a human programmer. Without the ability to fully decompose a CAD file into a matrix defining features and their corresponding attributes, there is no method—whether driven by a rule-based or ML system—that allows for automatic selection of tools and creation of operations. A robust feature recognition capability would serve not only as a backbone to programming automation, but it would also greatly assist in other analyses performed by machine shops, such as quoting prices and machining cycle times.

Consequently, feature recognition should be a near-term focus of those seeking to build an intelligent programming automation solution. While some solutions exist on the market and in academic research (see Section 1.5), no method sophisticated enough to replicate a human programmer cognitive process exists. Given the lack of success in creating a complete solution with rule-based approaches, recent advances

in deep learning might be the most promising path forward, especially given recent advances in computer vision in a variety of applications.

5.2 Other AI/ML Methods for Consideration

DLATS is one method of utilizing ML to automate programming: tool selection for hole drilling operations. However, there are a number of additional approaches that might have greater success; some of these are listed below.

- Tool selection variations. DLATS was chosen because tool selection is an integral part to the programming process. However, it was limited to only hole drilling operations; expansion of the training dataset to include other operation types may improve the accuracy with which tools can be selected. Furthermore, it is possible that an encoding of library reference numbers—rather than tool parameters formatted into a vector—can improve accuracy, given sufficient data.

- Prediction of operation parameters. Operation parameters can be predicted in a multitude of ways. The feed, speed, and other parameters dictating machine movement can be associated to feature geometry and tool data such that said parameters can be regressed from a predictive model. Additionally, a set of common operation *templates*, which contain specified operation parameters, can be created and a predictive algorithm can select from among that set. For a useful programming automation solution, a routine to predict operation parameters must be developed. Indeed, the research by Sharmaa, Chawlaa, and Rama demonstrated a method to predict G-code for simple drilling operations with up to 97% accuracy [29, 30].

- NLP to create G-code. Given the recent advances in natural language processing (NLP) to create human language models such as GPT-3 [42], it is feasible that G-code, or its associated .cls file, can be predicted directly from the CAD

.stp file. G-code, .cls files, and .stp files are simply line-by-line instructions specifying either the machine parameters or feature geometries. Human language models might inspire a method to consider these instructions as a language to serve as a basis for some NLP technique.

- Program optimization. A significant portion of the NRE process in program development involves iterative trials to ensure a sufficiently efficient process. As a result, some optimization routine to analyze created programs and associated machine performance to select the optimal tool and cutting conditions could dramatically save effective programming time. Such a routine could interface with a programming automation solution, improving the automated, suboptimal program.

Bibliography

[1] *F-35 Global Partnership | Lockheed Martin*. URL: https://www.lockheedmartin.com/en-us/products/f-35/f-35-global-partnership.html.

[2] William Pease. "An Automatic Machine Tool". In: *Scientific American* 187.3 (1952). ISSN: 0036-8733. DOI: 10.1038/scientificamerican0952-101.

[3] E. C. Johnson. "A numerically controlled cam-milling machine". In: *IRE Transactions on Industrial Electronics* PGIE-3 (1956). ISSN: 01975706. DOI: 10.1109/IRE-IE.1956.5007769.

[4] MMM Sarcar, K Mallikarjuna Rao, and K Lalit Narayan. *Computer aided design and manufacturing*. PHI Learning Pvt. Ltd., 2008.

[5] Michael J. Pratt. "Introduction to iso 10303—the step standard for product data exchange". In: *Journal of Computing and Information Science in Engineering* 1.1 (2001). ISSN: 15309827. DOI: 10.1115/1.1354995.

[6] *Drill Terminology and Cutting Characteristics | MITSUBISHI MATERIALS CORPORATION*. URL: http://www.mitsubishicarbide.com/en/technical_information/tec_rotating_tools/drills/tec_drills_technical_top/tec_drilling_terminology.

[7] Arivazhagan Anbalagan and Carlos Francisco Moreno-Garcia. "An IoT based industry 4.0 architecture for integration of design and manufacturing systems". In: *Materials Today: Proceedings*. Vol. 46. 2021. DOI: 10.1016/j.matpr.2020.11.196.

[8] ASME Y14.5-2009. "Dimensioning and Tolerancing Engineering Product Definition and Related Documentation Practices". In: *The American Society of Mechanical Engineers* (2009).

[9] *Geometric Manufacturing View Library*. URL: https://feature.geometricglobal.com/feature-recognition-suite/manufacturing-view-library/.

[10] *Feature Recognition & Grouping in Feature Based Machining | NX CAM*. URL: https://www.swooshtech.com/2020/03/06/feature-based-machining/.

[11] *CATIA Part Design: Using Feature Recognition (Manual) - IME Wiki*. URL: https://wiki.cadcam.com.my/knowledgebase/catia-part-design-using-feature-recognition-manual/.

[12] *Autodesk App Exchange (Subscription): Feature Recognition - IMAGINiT Manufacturing Solutions Blog*. URL: https://blogs.rand.com/manufacturing/2013/09/autodesk-app-exchange-subscription-feature-recognition.html.

[13] *Overview of FeatureWorks - 2019 - SOLIDWORKS Help*. URL: https://help.solidworks.com/2019/english/SolidWorks/fworks/c_Overview_of_FeatureWorks.htm?id=d5b3263dc5934bceb5f8b114be6e3723#Pg0.

[14] *About Feature Recognition*. URL: http://support.ptc.com/help/creo/creo_pma/usascii/index.html#page/part_modeling%2Ffeature_recognition_tool%2FAbout_Feature_Recognition.html%23.

[15] Liang Guo et al. "A hybrid 3D feature recognition method based on rule and graph". In: *International Journal of Computer Integrated Manufacturing* 34.3 (2021). ISSN: 13623052. DOI: 10.1080/0951192X.2020.1858507.

[16] Natarajan Krishnan and Gokulachandran J. "Application of Artificial Neural Network Techniques in Computer Aided Process planning- A review". In: *International Journal of Process Management and Benchmarking* 1.1 (2020). ISSN: 1460-6739. DOI: 10.1504/ijpmb.2020.10021148.

[17] Zhibo Zhang, Prakhar Jaiswal, and Rahul Rai. "FeatureNet: Machining feature recognition based on 3D Convolution Neural Network". In: *CAD Computer Aided Design* 101 (2018). ISSN: 00104485. DOI: 10.1016/j.cad.2018.03.006.

[18] Changmo Yeo et al. "Machining feature recognition based on deep neural networks to support tight integration with 3D CAD systems". In: *Scientific Reports* 11.1 (2021). ISSN: 20452322. DOI: 10.1038/s41598-021-01313-3.

[19] J. J. Shah and M. T. Rogers. "Expert form feature modelling shell". In: *Computer-Aided Design* 20.9 (1988). ISSN: 00104485. DOI: 10.1016/0010-4485(88)90041-3.

[20] Jung Hyun Han, Mike Pratt, and William C. Regli. "Manufacturing feature recognition from solid models: A status report". In: *IEEE Transactions on Robotics and Automation* 16.6 (2000). ISSN: 1042296X. DOI: 10.1109/70.897789.

[21] Bor Tyng Sheen and Chun Fong You. "Machining feature recognition and toolpath generation for 3-axis CNC milling". In: *CAD Computer Aided Design* 38.6 (2006). ISSN: 00104485. DOI: 10.1016/j.cad.2005.05.003.

[22] Jung Hyun Han et al. "Manufacturing feature recognition toward integration with process planning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 31.3 (2001). ISSN: 10834419. DOI: 10.1109/3477.931522.

[23] *Feature Based Machining: Introduction*. URL: https://community.sw.siemens.com/s/article/feature-based-machining-introduction.

[24] Kaarel Kruuser et al. “Implementation of a knowledge-based manufacturing on the example of sumar tools oü”. In: *Proceedings of the Estonian Academy of Sciences* 68.4 (2019). ISSN: 17367530. DOI: 10.3176/proc.2019.4.10.

[25] *SpeedMill - Automation of CAM programming | JANUS Engineering Switzerland*. URL: https://www.janus-engineering.com/ch_en/solutions/cam-computer-aided-manufacturing/speedmill/.

[26] Dong Hyeon Kim et al. *Smart Machining Process Using Machine Learning: A Review and Perspective on Machining Industry*. 2018. DOI: 10.1007/s40684-018-0057-y.

[27] S. Klancnik, M. Brezocnik, and J. Balic. “Intelligent CAD/CAM system for programming of CNC machine tools”. In: *International Journal of Simulation Modelling* 15.1 (2016). ISSN: 17264529. DOI: 10.2507/IJSIMM15(1)9.330.

[28] Marc André Dittrich, Florian Uhlich, and Berend Denkena. “Self-optimizing tool path generation for 5-axis machining processes”. In: *CIRP Journal of Manufacturing Science and Technology* 24 (2019). ISSN: 17555817. DOI: 10.1016/j.cirpj.2018.11.005.

[29] Neelima Sharma, V. K. Chawla, and N. Ram. “Comparison of machine learning algorithms for the automatic programming of computer numerical control machine”. In: *International Journal of Data and Network Science* 4.1 (2020). ISSN: 25618156. DOI: 10.5267/j.ijdns.2019.9.003.

[30] Neelima and Vivek Chawla. “Automated CNC Programming by the Restricted Boltzmann Machine Algorithm”. In: *Lecture Notes in Mechanical Engineering*. 2021. DOI: 10.1007/978-981-15-5463-6{_}62.

[31] Dheeraj Peddireddy et al. “Deep learning based approach for identifying conventional machining processes from CAD data”. In: vol. 48. 2020. DOI: 10.1016/j.promfg.2020.05.130.

[32] Randall Davis, Bruce Buchanan, and Edward Shortliffe. “Production rules as a representation for a knowledge-based consultation program”. In: *Artificial Intelligence* 8.1 (1977). ISSN: 00043702. DOI: 10.1016/0004-3702(77)90003-0.

[33] Brent Smith and Greg Linden. “Two Decades of Recommender Systems at Amazon.com”. In: *IEEE Internet Computing* 21.3 (2017). ISSN: 10897801. DOI: 10.1109/MIC.2017.72.

[34] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015). ISSN: 10959203. DOI: 10.1126/science.aab3050.

[35] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information Fusion* 58 (2020). ISSN: 15662535. DOI: 10.1016/j.inffus.2019.12.012.

[36] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. 2014.

[37] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June-2015. 2015. DOI: 10.1109/CVPR.2015.7298640.

[38] M. Nadeem Ahangar et al. *A survey of autonomous vehicles: Enabling communication technologies and challenges*. 2021. DOI: 10.3390/s21030706.

[39] *Building The Automotive Workforce For The Future*. URL: https://www.oliverwyman.com/our-expertise/insights/2018/sep/automotive-manager-2018/resources/building-the-automotive-workforce-for-the-future.html.

[40] *Amesite » How Will Sweeping Changes in Automotive Products Affect Automotive Workforces?* URL: https://amesite.com/blogs/how-will-sweeping-changes-in-automotive-products-affect-automotive-workforces/.

[41] *Toyota and Lexus Are Finally Catching up to Tesla in Level 2*. URL: https://www.motorbiscuit.com/toyota-lexus-finally-catching-tesla-level-2/.

[42] Luciano Floridi and Massimo Chiriatti. *GPT-3: Its Nature, Scope, Limits, and Consequences*. 2020. DOI: 10.1007/s11023-020-09548-1.