

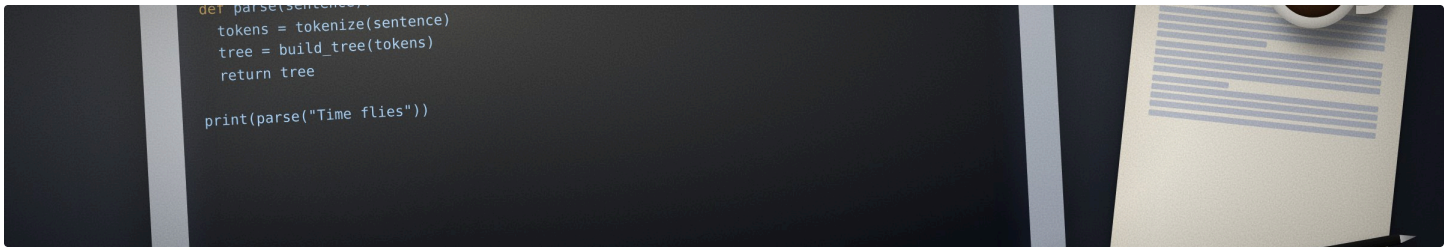
ADMISSIONS STRATEGY

Computational vs. Theoretical: Why Applying to the Wrong Department Subfield Can Weaken Your Application



By **Alex Gomelskiy**
Founder and Director of Admissions Strategy, Scholar Coach, Inc.

October 20, 2026



A student builds an impressive application: several programming projects, a polished GitHub portfolio, perhaps an internship or a research experience involving artificial intelligence. Yet when asked why they want to study computer science at a particular university, their answer focuses almost entirely on building software—even though the program's curriculum places substantial emphasis on mathematical foundations, algorithms, and the theory of computation.

The problem is not that software development is less intellectually valuable than theoretical computer science. It is that the student may not have investigated what studying computer science at that institution actually entails.

Computer science is not a single academic pursuit. It encompasses questions about what computers can calculate, how efficiently they can solve problems, how software should be designed, how operating systems manage resources, and how machine learning models draw conclusions from data. Students who understand these distinctions can build more credible applications by connecting their interests to the specific educational opportunities a university provides.

The goal is not to guess which subfield an admissions committee prefers. It is to demonstrate that the student understands the program they are applying to and has a genuine reason to pursue it.

1. COMPUTER SCIENCE HAS MORE THAN ONE INTELLECTUAL CENTER

Families often evaluate computer science programs by looking at reputation, rankings, internship opportunities, and the technologies students learn. Those factors can matter, but they do not necessarily reveal the intellectual character of a curriculum.

Consider two broad areas of computer science.

Theoretical computer science investigates the mathematical foundations of computation. Students may study algorithm design, computational complexity, formal logic, cryptography, and the limits of what can be computed efficiently. The central questions are not simply how to write a program, but why an algorithm works, how its performance can be established, and whether a problem can be solved within practical constraints.

Applied and systems-oriented computing focuses more directly on designing, implementing, testing, and evaluating computational systems. Depending on the program, students may investigate software architecture, operating systems, databases, networks, human-computer interaction, or the engineering challenges involved in deploying reliable software.

These categories overlap substantially. Algorithm analysis informs practical software development, while systems engineering often requires sophisticated mathematical reasoning. Artificial intelligence also draws on both

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →

theoretical foundations and applied implementation.

Stanford University's computer science department illustrates this breadth through its undergraduate tracks, which include systems, theory, artificial intelligence, and other areas. Its theory track covers topics such as algorithms, logic, formal methods, and computational limitations, while its systems track includes operating systems, compilers, networks, and large-scale systems.

The implication for applicants is straightforward: a strong programming portfolio demonstrates one kind of preparation, but it does not automatically communicate an interest in every dimension of computer science. Students need to identify which questions genuinely motivate them and understand how those questions connect to a university's curriculum.

Source: Stanford University, Computer Science Bachelor's Program.

2. READ THE REQUIRED COURSES BEFORE CHOOSING YOUR APPLICATION NARRATIVE

A department's introductory course requirements provide a more reliable starting point than its promotional language or a list of popular electives.

Families should examine three categories of courses:

Mathematical foundations: discrete mathematics, probability, proofs, and linear algebra can indicate the mathematical preparation expected for later work.

Core computer science: programming, data structures, algorithms, and computer systems establish the foundational knowledge students are expected to develop.

Advanced electives and research: courses in theory, software engineering, artificial intelligence, security, or human-computer interaction reveal possible directions for deeper study.

No single category tells the whole story. The important question is how the requirements fit together.

MIT's Bachelor of Science in Computer Science and Engineering, for example, includes programming, software construction, mathematics for computer science, algorithms, and computer systems. Students also encounter subjects such as computability and complexity theory or advanced algorithm design, alongside opportunities to explore other areas of computing.

This curriculum does not divide students neatly into programmers and mathematicians. Instead, it combines implementation with analytical foundations. An applicant who describes a fascination with building applications could make a credible connection to software construction and systems engineering. Another student might be more interested in proving algorithmic properties, understanding computational limits, or exploring the mathematics behind machine learning.

Both interests can fit the same degree. The difference is which parts of the curriculum provide the most meaningful direction for each student.

Sources: MIT EECS undergraduate curriculum and MIT Computer Science and Engineering degree requirements.

3. MATCH THE ACADEMIC QUESTIONS TO THE EVIDENCE IN YOUR APPLICATION

Students often describe their interests through the products they have built rather than the questions that motivated them.

"I developed an application that helps students organize assignments" describes an accomplishment. It does not yet explain the student's intellectual interests.

A more developed account might explain that the student became interested in how software can remain reliable as the number of users grows, how data should be structured to support fast retrieval, or how interface design affects whether people can use a tool effectively.

Similarly, "I competed in a programming competition" identifies an experience, but the intellectual substance may lie in the student's interest in algorithmic efficiency, problem decomposition, or mathematical reasoning.

The objective is not to make every project sound theoretical. It is to articulate the thinking behind the work.

Consider how the same general interest can lead to different academic directions:

A student interested in artificial intelligence might focus on training and evaluating models, investigate the mathematical principles behind learning algorithms, or explore the engineering required to deploy a model reliably. Each direction raises different questions and may connect to different courses or research groups.

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →

A student interested in cybersecurity might investigate cryptographic proofs, the implementation of secure systems, or the human behavior that creates vulnerabilities. These interests overlap, but they are not interchangeable.

A student interested in building software products might explore software architecture, testing, distributed systems, or human-computer interaction. A compelling application can explain why the student wants to understand not only how to build a product, but also how to make it reliable, maintainable, efficient, or accessible.

The strongest narrative emerges when the student's previous work, current questions, and prospective academic study form a coherent connection. That coherence should come from genuine experience, not from retroactively assigning an impressive-sounding specialization to an ordinary project.

4. DO NOT CONFUSE CURRICULUM FIT WITH ADMISSIONS RULES

There is an important distinction between choosing an academic direction that fits and applying to a university that admits students directly into a particular program.

Some universities admit students to a university or undergraduate college before they declare a major. Others have direct-admission programs, separate schools, or additional requirements for particular majors. Policies can differ even within the same institution.

Princeton University, for example, states that students apply to Princeton University rather than to individual departments, programs, or schools. Its computer science department offers both Bachelor of Arts and Bachelor of Science in Engineering degree paths, with different general education and degree requirements.

That structure matters. A student should not assume that writing about theoretical computer science will improve admission odds simply because Princeton offers theoretical courses. The more defensible objective is to explain an authentic academic interest within the university's broader educational environment.

At institutions that do admit students directly into a specific major or program, the intended field may have a more direct role in the application process. Even then,

families should consult the institution's official admissions guidance rather than infer selection criteria from course titles alone.

A mismatch can make an application less convincing, but it does not establish why an applicant will be rejected. Admissions decisions involve multiple factors, and an imperfectly tailored essay is not proof that a student chose the wrong subfield.

Source: Princeton University, Academics and Admissions.

5. A PRACTICAL CURRICULUM-FIT AUDIT FOR FAMILIES

Before finalizing a college list or writing a major-specific essay, students can conduct a short review of each program.

Step 1: Identify the actual degree and admissions pathway. Confirm the degree name, the school or college administering it, whether students apply directly to the major, and whether internal transfer restrictions apply.

Step 2: Review the required courses. Start with introductory programming, mathematics, algorithms, and systems requirements. Then examine the courses students must take beyond the introductory sequence. Requirements matter more than an appealing elective that only a small number of students may take.

Step 3: Identify two or three relevant advanced opportunities. Look for courses, laboratories, faculty research, or undergraduate projects connected to the student's real interests. Verify that the opportunities are available to undergraduates and determine whether prerequisites or enrollment restrictions apply.

Step 4: Compare the curriculum with the student's existing work. Ask what the student actually enjoyed: debugging a complex system, proving an algorithm's correctness, analyzing data, studying model behavior, or designing an interface. The answer should guide the academic narrative rather than the other way around.

Step 5: Test the explanation. Can the student explain why a particular course, research opportunity, or curricular feature matters to a question they already care about? If the answer is merely that the university is prestigious or the course sounds advanced, more investigation is needed.

The purpose of this audit is not to find a perfect match between every high school activity and a college course. It

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →

is to ensure that the student understands the educational path they are proposing to pursue.

HOW SCHOLAR COACH HELPS STUDENTS ESTABLISH ACADEMIC FIT

Scholar Coach helps students develop college applications around their actual intellectual interests, previous work, and intended direction of study.

Map the student's technical interests. We examine projects, coursework, competitions, research, and independent learning to identify the questions the student genuinely wants to explore—not simply the technologies they have used.

Investigate each program's curriculum. We review official degree requirements, relevant advanced courses, undergraduate research opportunities, and admissions pathways to distinguish meaningful differences between universities.

Connect experience to academic direction. We help students explain what their work taught them, which problems remain unresolved, and why further study in a

particular area makes sense.

Develop institution-specific essays. Rather than repeating generic claims about innovation or changing the world through technology, students build explanations grounded in verified academic opportunities and their own experiences.

Check coherence and accuracy. We review whether the application accurately represents the program, whether the student's stated interests align with the evidence presented, and whether claims about departmental admissions or research access are supported.

A student does not need to arrive in college with a permanently fixed specialization. Computer science is broad precisely because its fields inform one another. What matters is demonstrating thoughtful preparation, a credible reason for pursuing the discipline, and an understanding of how the university can support the student's next stage of learning.

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →