I want to do research in topics like programming languages, compilers, and program verification. Through my undergraduate studies, I have gained a solid foundation across disciplines, thanks to my comprehensive curriculum in Computer Science, Electrical and Computer Engineering, and Physics. These diverse areas of study have equipped me with the skills to approach problems from different perspectives. My exploratory projects, such as designing a language framework for typesetting systems and proposing an augmented type system for JavaScript, alongside projects in operating systems and AI, like building a mini kernel and an AI-based storyteller, have deepened my interests and prepared me with the necessary skill set. With a strong interest in further advancing the field of programming languages, I am applying for graduate study to pursue my academic and research goals.

During my undergraduate years, a comprehensive curriculum provided me with both a broad academic foundation and advanced coursework in programming languages, compilers, and systems. In Programming Language Design (CS 422), I created languages in different paradigms using the K framework. In Compiler Construction (CS 426), I developed a functional KOOL compiler, and in Operating System Design (CS 423), I built several practical Linux kernel modules. These projects provided me with a strong understanding of language design, compiler construction, and operating systems. Furthermore, my coursework in Digital Systems (ECE 385) and Computational Physics (PHYS 498) strengthened my analytical and hardware-level thinking, complementing my perspective on systems and computation. Together, these courses have equipped me with the theoretical foundation and practical skills necessary to pursue innovative research in programming languages and compilers.

Building on the foundational knowledge from my coursework in programming languages and compilers, I began an exploratory project to address challenges I encountered with LaTeX during technical writing. The idea of creating a typesetting system to replace LaTeX emerged from my frustration with its poor readability and limited flexibility when writing complex ODE formulas, big-step semantics, and TikZ graphs. Lines of dense macros made source files unreadable, while simple adjustments, like spacing, often required opaque “black magic.”

To address these issues, I began designing a new language that improves LaTeX's readability and expandability. I studied The TeXbook and LuaTeX documentation. It introduces some effective typesetting concepts like TeX's Glue-Box model but relies on an outdated macro system and lacks modern Unicode support. Based on these insights, I designed a Lisp-like domain-specific language, which allows user-defined syntax and semantics to capture frequent constructs, such as those in ODE formulas or big-step semantics. This could fold cumbersome expressions into human-readable syntax, making the language significantly more readable and usable. I also prototyped an IR that generates instructions for rendering in browsers or PDFs according to OpenType font specification. While the project is still preliminary, it

has strengthened my understanding of modular language design and broadened my perspective on translating high-level constructs into graphical outputs.

Beyond the typesetting project, I also explored language design through a proposal to address limitations I observed in TypeScript, a typed superset of JavaScript that I frequently use in web programming. While its subtyping system shows promise for refined typing that could better capture the behavior of JavaScript APIs, the lack of type operations often leads to convoluted "type gymnastics" or manual workarounds, undermining its expressiveness.

To address these limitations, I have made a proposal for a predicate-based type system. Compile-time predicate functions would be defined to check type constraints based on metadata of the checked variables. Certain type operations could then be defined in predicate functions, whereas commonly used types could be treated as syntactic sugar, making the language more flexible. It is consistent with imperative paradigms in JavaScript, where TypeScript's system is often inconsistent. The proposal have deepened my understanding of predicate-based systems and how to apply it to practical type system design.

In addition to my major projects, I have worked on several smaller projects that further developed my technical skills. I have built an AI-based interactive storyteller that generates story pieces based on user prompts. In a course project, I implemented a mini kernel for x86 architecture, which deepened my understanding of operating systems and low-level programming. Additionally, I reconstructed the Nintendo NES console on FPGA as part of a digital systems course, applying hardware design principles to a real-world system. These projects complemented my academic and research preparation by broadening my perspective in areas in hardware, systems, and AI.

My coursework and projects have deepened my interest in programming languages and compilers, leading me to seek opportunities at UCSD to collaborate with experts in the field and contribute to its vibrant research environment. I am particularly interested in creating augmented type systems for practical programming languages, using verification techniques to enhance security and maintainability in software and systems. Additionally, I want to explore universal compiler generators and domain-specific language design, and simplify the language creation process while preserving performance. UCSD's strong curriculum and collaborative research environment make it the ideal place to pursue these goals.

I am particularly drawn to this opportunity to work with Professors Deian Stefan and Ranjit Jhala, whose research closely aligns with my academic interests. For example, the work of Professor Stefan on WaVe—a WebAssembly runtime with verifiable security—manifests the effective application of formal methods to systems design. His work on extensible effects and safety policies is especially very much in line with my interest in designing verifiable systems, which offer strong security

guarantees at the minimum complexity. Similarly, work that Professor Jhala has been carrying out on refinement types, and especially within the setting of Flux, demonstrates how type systems can be used to increase usability and correctness of practical programming languages like Rust. His approach to ergonomic verification through type-directed invariant synthesis inspires me to research how formal methods address the real-world challenges in programming language design. Working with these scholars at UCSD will provide an unparalleled opportunity to study how rigorous formalism bridges the gap between theory research and practical system development and to make meaningful contributions to the field