IGNITION PROTOCOL

AN EVOLUTIONARY ENGINE
FOR MOLECULAR COMBUSTION

STRIPE YOUNG SCIENTIST & TECHNOLOGY EXHIBITION
2026

by Finn Harnedy

A computational framework for discovering, filtering, and operationalising synthetic fuel candidates through molecular generation and blend optimization.

Table of Contents

Table of Contents
2. Introduction
3. Literature Review
  3.1 Historical Development of Gasoline and Diesel Fuels
  3.2 Combustion Science Fundamentals: Stoichiometry, Ignition, and Flame Speed
  3.3 Emissions Chemistry: NOx, Soot, CO, CO_2, and Aldehydes
  3.4 Regulatory Drivers: Emission Standards and Fuel Policy
  3.5 Fuel Performance Metrics: Octane, Cetane, and Other Key Indices
4. Computational Benchmarking of the Engine
  Engine Structure and Workflow
  Benchmark Profiles and Setup
    Ultra-snappy Profile
    Demo Profile
    Classroom Profile
    Light Stress Profile
    Heavy Stress Profile
    Brutal Profile
  Performance Comparison and Scaling Behavior
  Bottleneck Analysis
  Opportunities for Optimization and Future Integration
5. Implemented System Architecture
  5.1. Major modules
    A. Molecular generation and evaluation
    B. Blend optimisation engine
    C. Command line orchestration and run management
  5.2. End to end data flow
  5.3. Determinism and reproducibility
6. Methodology
  6.1. Fragment preparation and cleaning
  6.2. Evolutionary generator and molecular construction
  6.3. Fitness evaluation
  6.4. Bridging molecules to blend components
  6.5. Deterministic blend optimisation
  6.6. Planned experimental validation
  6.7. Methodological Scope and Limitations
    6.7.1. Absence of Experimental and Wet-Laboratory Validation
    6.7.2. Model-Based Property Estimation and Idealised Mixing Assumptions
    6.7.3. Fragment-Constrained Chemical Space Exploration
    6.7.4. Feasibility and Regulatory Proxy Constraints
    6.7.5. Instrumentation Guidance and Deferred Experimental Implementation
    6.7.6. Laboratory Access Outreach at University College Cork
    6.7.7. Reproducibility Within Declared Computational Scope
7. Results and Preliminary Outputs
  7.1. Basic Engine Results
    7.1.1. Starting Molecule Pool and Fragment Diversity
    7.1.2. Evolved Molecules After 20 Generations
    7.1.3. Fitness Progression Across Generations
  7.2. Unified Pipeline Results
    7.2.1. Blend optimisation outputs - Top-N Ranked Blends
      7.2.2. Blend Optimisation: Portfolio Diversity Analysis
      7.2.3. Blend Optimisation: Component Library Data
      7.2.4. Blend Optimisation: Run Metadate (RUN>json)
    7.2.5. Molecular generation outputs
8. Discussion

9. Conclusion
References

1.Abstract

Ignition Protocol is a computational framework for discovering, filtering, and operationalising synthetic fuel candidates. In the current implementation, the system is a unified pipeline with two coupled engines:

(1) a molecular generation module that evolves candidate molecules from fragment libraries using RDKit-based validity checks and a multi-objective fitness score, and

(2) a deterministic blend optimisation engine that selects and proportions real components (including optional “novel” candidates) into lab-ready fuel blends subject to fuel-type specifications.

The molecular generator produces candidate SMILES strings and attaches practical metadata (estimated fuel properties, synthesis feasibility and a suggested synthesis route, plus simple storage-stability proxies). The blend optimiser converts a component library into an optimised recipe (volume fractions, mL and grams per 1000 mL) while enforcing constraints such as volatility limits (T10/T90, RVP), minimum octane/cetane requirements, density bounds (diesel/jet), and caps on each component’s maximum volume fraction.

The project is designed to bridge computational fuel design with proposed experimental validation: combustion “fingerprinting” (flame imaging and spectroscopy) and environmental stability assays (storage stability and biodegradability screening). Where experimental procedures are described in this manuscript, they are presented as a planned validation framework.

2.Introduction

Synthetic fuels and engineered fuel blends offer a pathway to reducing reliance on fossil energy while continuing to meet global demand under increasingly stringent emissions constraints. Conventional fuel formulation strategies largely focus on incremental modification of known components, which limits exploration of the broader chemical space of fuel-like molecules and formulations.

Recent advances in computational chemistry and optimisation enable far wider exploration of this space. However, many molecular design approaches stop at ranking abstract candidates and do not produce outputs that are directly compatible with real fuel specifications or experimental workflows.

Ignition Protocol addresses this gap by integrating computational molecular design with practical blend optimisation. The project combines a fragment-based evolutionary generator for proposing fuel-like molecules with a deterministic blend optimisation engine that enforces real-world fuel constraints and produces laboratory-ready mixing recipes. Rather than optimising molecules in isolation, the system focuses on generating actionable fuel formulations that can be reproduced, tested, and iterated upon.

This report presents the design, implementation, and evaluation of the Ignition Protocol pipeline. Following a review of relevant combustion science, fuel metrics, and prior computational approaches, the system architecture and performance are characterised. Computational optimisation results and preliminary outputs are then reported, alongside a structured plan for experimental validation.

3.Literature Review

In this section, I review the scientific and engineering foundations relevant to the Ignition Protocol, an AI-driven engine for novel fuel discovery. I cover the historical development of fuels, fundamentals of combustion and emissions chemistry, regulatory drivers, key fuel performance metrics, advanced engine combustion modes, and the state-of-the-art in applying artificial intelligence to fuel design. I also discuss experimental combustion diagnostics, environmental considerations, synthesis feasibility, and broader policy context. Throughout, I connect these topics to justify the approach taken in my project. I write from a first-person perspective, reflecting my understanding of the literature and how it guides my research.

3.1 Historical Development of Gasoline and Diesel Fuels

Early Gasoline and Internal Combustion Engines: Gasoline emerged in the late 19th century as a volatile petroleum byproduct that found use in newly invented internal combustion engines. Early gasoline was essentially a surplus from kerosene refining (kerosene was used for lamps), but innovators like Nikolaus Otto and Gottlieb Daimler recognized that these light hydrocarbons could fuel efficient spark-ignition engines. By the 1880s–1890s, gasoline-fueled engines powered the first automobiles. Initially, fuel quality was inconsistent; engines had low compression and very basic carburetion, tolerating the available gasoline. Over time, refining processes improved to increase gasoline's quality and consistency, guided by growing understanding of fuel chemistry and engine requirements. Rudolf Diesel and the Birth of Diesel Fuel: In 1897, Rudolf Diesel demonstrated the first compression-ignition engine, initially using peanut oil as fuel. This engine relied on auto-ignition of the fuel under high pressure rather than a spark. As diesel engines became commercial (early 20th century), the petroleum industry introduced diesel fuel, a heavier distillate than gasoline. Diesel fuel's properties differed significantly: it needed to ignite quickly upon injection rather than resist ignition. Early diesel fuels were not highly refined – often just a fraction of crude oil – but over decades fuel standards improved to ensure sufficient ignition quality (cetane) and consistent performance. The concept of cetane number (ignition quality rating) was developed in the 1930s to quantify how readily a diesel fuel ignites (analogous to octane rating for gasoline, but essentially the inverse property). I discuss octane and cetane in detail in Section 3.5. Tetraethyl Lead and Octane Enhancement: A major historical milestone was the introduction of tetraethyl lead (TEL) in the 1920s as an anti-knock additive for gasoline. As engines in the 1910s–20s began to use higher compression ratios for efficiency, knocking (uncontrolled combustion) became a serious problem. In 1921, Thomas Midgley discovered that adding minute amounts of TEL dramatically raised the fuel's octane rating (knock resistance). For decades, leaded gasoline became the norm worldwide, enabling more powerful engines. However, by the 1970s the severe toxicity and environmental pollution from lead became clear. This led to one of the most significant shifts in fuel policy: unleaded gasoline was introduced and progressively mandated. In the United States, the 1970 Clean Air Act and subsequent regulations forced the phase-out of lead, primarily because catalytic converters (required to reduce emissions) are poisoned by lead deposits. By the mid-1980s, leaded fuel had been largely eliminated in the U.S. (and later in Europe and other regions by the 1990s) in favor of unleaded, along with a reduction in compression ratios or the use of alternative octane boosters. Emergence of High-Octane Blending Components: Post-LEAD era, refineries began producing higher-octane gasoline through reforming processes (to create aromatics) and used alternative additives. For example, MTBE (methyl tert-butyl ether) and later ethanol were used as octane boosters in the 1980s–2000s, which also had the benefit of adding oxygen to fuel (helping reduce carbon monoxide emissions). These changes were driven by both performance needs and environmental regulations (as discussed later in Section 3.4). Historical Diesel Fuel Developments: Diesel fuel quality improvements historically focused on cold-weather performance and cetane. In early years, diesel engines could be finicky, with hard starting and smoke. Over time, refiners increased the cetane number by improving distillation and incorporating additives (e.g. alkyl nitrates) to ensure diesel ignites promptly in engines. Another historical issue was high sulfur content in both gasoline and diesel; sulfur contributes to engine deposits and, when combusted, forms SOx emissions. We later reduced sulfur drastically (toward ultra-low sulfur diesel (ULSD) and low-sulfur gasoline) as part of emissions control programs in the 2000s. This shift is a crucial enabler for catalytic aftertreatment systems and is discussed further under regulations. In summary, the early development of automotive fuels was a process of trial-and-error coupled with scientific progress in chemistry and engine engineering. By the mid-20th century, fuel formulations had evolved (e.g. leaded high-octane gasoline and adequate cetane diesel) to meet performance demands of the day. However, the latter half of the 20th century saw a new driver for fuel innovation: air pollution and public health, which led to cleaner fuels and new fuel blends. The next sections will cover how combustion chemistry and emission concerns necessitated these changes, and how regulatory frameworks pushed fuel technology forward.

3.2 Combustion Science Fundamentals: Stoichiometry, Ignition, and Flame Speed

To design and evaluate fuels, you must understand basic combustion science. Here I outline key concepts such as the stoichiometric mixture, ignition behavior, and flame propagation – all of which influence engine performance and emissions. Stoichiometric Air-Fuel Ratio: Every fuel has a stoichiometric air-fuel ratio (AFR), which is the precise mass ratio of air to fuel needed for complete combustion (with no leftover oxygen or fuel). This ratio depends on the fuel's elemental makeup (carbon, hydrogen, oxygen content). For example, gasoline (approximated as C_7H_16) has a stoichiometric AFR of about 14.7:1 (meaning ~14.7 kg of air per 1 kg of fuel). Diesel fuel, with a similar C/H ratio, is around 14.5:1. Oxygenated fuels have a lower stoichiometric AFR because the fuel itself contributes oxygen. Ethanol (C_2H_5OH), for instance, has a stoichiometric AFR near 9:1, much lower than gasoline. This is important because it affects how the fuel burns in engines. In a spark-ignition engine, the mixture is often controlled at or near stoichiometric for best performance and to allow catalytic converters to function (they need a roughly stoichiometric exhaust). If the mixture is rich (too little air), combustion leaves fuel unoxidized, causing carbon monoxide (CO) and unburned hydrocarbon emissions. If the mixture is lean (excess air), combustion can be more complete (less CO/UHC), but very lean mixtures can misfire or, in some cases, lead to high nitrogen oxides due to high combustion temperatures. The presence of oxygen in the fuel can help ensure more complete combustion in situations like cold starts, where engines deliberately run rich. Historically, additives like MTBE and ethanol were added to gasoline in the 1990s to oxygenate fuel and reduce CO emissions in cities. In my project, I consider molecules with built-in oxygen (like alcohol functional groups) beneficial from an emissions standpoint: oxygenated fuels "carry their own oxygen", potentially reducing incomplete combustion by enabling leaner combustion or by oxidizing soot precursors internally. However, as noted, oxygenated fuels also tend to have lower energy content (because the oxygenated part of the molecule does not release energy), so there is a trade-off addressed later in Section 3.6. Ignition Delay and Auto-Ignition: Ignition behavior is crucial for engine compatibility. In spark-ignition (SI) engines (gasoline), we actually want to avoid auto-ignition of the fuel-air mixture until the spark plug fires – this property is quantified by octane number (discussed later). In compression-ignition (CI) engines (diesel), we rely on auto-ignition of fuel injected into hot air – quantified by cetane number. The time between fuel injection and the start of combustion in a diesel is called ignition delay. A short ignition delay (high cetane fuel) results in a smoother, controlled burn; a long delay (low cetane) leads to a larger amount of premixed fuel igniting at once, causing a violent pressure spike known as "diesel knock" and high NOx formation. From a fundamentals perspective, auto-ignition chemistry is governed by the fuel's molecular structure and the conditions (temperature, pressure). Straight-chain alkanes ignite easily (why hexadecane was defined as cetane = 100), whereas highly branched or aromatic molecules resist ignition. I leverage these principles: for instance, if I design a fuel for a diesel application, I would seek molecular features that promote quick ignition (linear alkyl chains, perhaps with nitrate functional groups as in some cetane improvers). For spark-ignition fuels, I want the opposite: molecules that are resistant to ignition (branched chains, aromatics, certain ethers) to prevent premature knock. The detailed kinetics of auto-ignition are complex, involving low-temperature oxidation pathways and radical chain reactions, but a basic understanding is captured by octane/cetane ratings used in engine tests. Laminar Flame Speed: Once ignition occurs (either by spark or auto-ignition), the flame propagates through the fuel-air mixture. The laminar flame speed is the speed at which an unstretched flame front travels through a quiescent premixed gas. It is a fundamental property of the fuel-air mixture. Typical laminar flame speeds for stoichiometric gasoline-air are on the order of 30–40 cm/s. Hydrogen has an exceptionally high flame speed (~200 cm/s), while heavier hydrocarbons and especially fuels with high aromatic content burn slower. In an engine, faster flame speed (up to a point) is beneficial because it means the combustion can complete closer to the optimal timing (around top-dead-center of the piston). If a fuel's flame speed is too slow, combustion may continue late into the expansion stroke, causing a loss of efficiency and more unburnt fuel emissions. Very slow combustion can also force designers to run engines at less aggressive conditions (to avoid misfire or knock onset due to partial burns). If flame speed is extremely high (approaching a detonation), that's essentially knock – which we avoid in SI engines. In my literature review, I found that flame speed correlates with molecular structure: fuels with simpler, reactive structures (like small olefins or highly branched alkanes) tend to burn faster than large aromatics. Thus, if designing fuels for lean or high-EGR (exhaust gas recirculation) combustion strategies, one might favor components with higher flame speeds to sustain stable combustion in dilute mixtures. Modern researchers often calculate or measure laminar flame speeds for novel fuel candidates to ensure they will combust rapidly enough in an engine. Combustion Temperature and Flame Chemistry: Another fundamental aspect is flame temperature, which is mainly a function of the fuel's heat release and mixture stoichiometry. Stoichiometric combustion of most hydrocarbon fuels yields flame temperatures around 2000–2300 K. Higher flame temperatures promote the formation of thermal NOx (discussed in Section 3.3). Fuels containing certain functional groups (like alcohols) can have slightly different adiabatic flame temperatures due to their oxygen content and lower heating value – often resulting in a cooler flame at stoichiometric conditions. Additionally, flame chemistry determines the production of intermediate radicals and soot precursors. Fuels that decompose into small fragments (like short-chain alkanes) tend to produce fewer sooting precursors than fuels that readily form ring structures (aromatics). These chemical tendencies will be explored more under emissions (Section 3.3). In summary, the basics of combustion science – the proportion of air needed, the ease of ignition, the speed of flame propagation, and the resultant temperatures – are all crucial for evaluating a potential fuel. As I design new molecules, I must consider how these properties will play out: e.g., will the molecule likely ignite too soon or not soon enough? Will it burn fast and clean, or slow and smokily? By grounding my approach in these fundamentals, I ensure the candidate fuels are not just numerically optimized by an algorithm, but also make sense from a combustion standpoint.

3.3 Emissions Chemistry: NOx, Soot, CO, CO_2, and Aldehydes

Burning fuel in engines produces a variety of chemical emissions. One of my core motivations for seeking new fuels was to reduce harmful emissions while maintaining performance. Here I review how different pollutants form during combustion and how fuel properties influence them. Carbon Dioxide (CO_2): CO_2 is the end-product of complete combustion of carbon in the fuel. While not toxic, CO_2 is the primary greenhouse gas driving climate change. The amount of CO_2 emitted is directly related to the fuel's carbon content and how much fuel is burned (thus indirectly related to engine efficiency). Fuel design alone can't eliminate CO_2 – any hydrocarbon or alcohol fuel will produce CO_2 if fully burned. However, fuels with higher hydrogen content (or with oxygen content enabling leaner burn) produce slightly less CO_2 per unit energy. For example, ethanol (C_2H_5OH) has a lower carbon-to-hydrogen ratio than octane (C_8H_{18}), so per unit energy, ethanol generates less CO_2 (some of the energy comes from forming water). Additionally, if a fuel allows higher efficiency (e.g., higher compression ratio due to high octane), the vehicle will burn less fuel for the same work, reducing net CO_2. In my project, I am not directly optimizing for CO_2 emissions (since that's largely an outcome of carbon content and efficiency), but I prioritize high energy density and engine efficiency which indirectly helps minimize CO_2 per km. Ultimately, addressing CO_2 may require using bio-derived or synthetic fuels made from CO_2 (so the carbon is recycled), but that's a broader life-cycle issue beyond the combustion chemistry focus of Ignition Protocol. Carbon Monoxide (CO) and Unburned Hydrocarbons (UHC): CO and UHC result from incomplete combustion. CO forms if there is insufficient oxygen or insufficient time to fully oxidize carbon to CO_2. UHC (which can be thought of as droplets or pockets of fuel that never fully burned, or intermediate hydrocarbons like methane, ethylene, etc.) occur for similar reasons. In gasoline SI engines, CO and UHC emissions are highest during rich operation (such as cold start or full throttle) when the mixture deliberately has excess fuel. Modern engines use catalytic converters to oxidize CO and UHC into CO_2 and water after combustion. Diesel engines typically run lean (excess oxygen), so they emit relatively low CO, but can still emit UHC (often visible as white vapor or smell of unburned diesel, especially at cold start or idle). Fuel properties influence these emissions: a fuel with high volatility and good mixability will vaporize and mix thoroughly, leaving fewer liquid droplets or pockets to burn incompletely. A fuel with oxygen content (like oxygenates) tends to produce less CO/UHC because even if the local mixture is rich, the fuel itself provides some oxygen to help complete the reaction. Historically, cities with CO pollution problems turned to oxygenated fuels; for example, adding 10% ethanol to gasoline can significantly cut CO emissions in older engines by leaning out the mixture and promoting complete oxidation. In my literature review, I found that the stoichiometric requirement plays a role: fuels that require a very rich mixture for stability will inevitably create more CO/UHC. Therefore, one aim in fuel design is to favor fuels that either allow leaner operation or inherently combust more completely. Also, reducing heavy high-boiling components in fuel can reduce UHC, because heavy molecules may survive combustion partially or burn slowly. New synthetic fuels that are tailored to have only easily vaporized, clean-burning components could drastically reduce engine-out CO and hydrocarbon emissions (aside from cold start, which is a special case needing engine strategies like heating or spark timing). Nitrogen Oxides (NOx): NOx (a mix of NO and NO_2) forms when nitrogen and oxygen in the air react at the high temperatures of combustion. In engines, the thermal NOx mechanism is dominant: above ~1800 K, atmospheric N_2 can oxidize to NO. Thus, peak combustion temperature and oxygen availability largely govern NOx formation. SI gasoline engines typically operate near stoichiometric and have flame temperatures around 2200 K, producing significant NOx, which is then cleaned by a three-way catalyst. Diesel engines run lean and at high compression, often producing even more NOx because of high combustion temperatures in localized zones. Fuel influences NOx in a few ways. First, any fuel property or engine condition that lowers peak temperatures will reduce NOx. For instance, high heat-of-vaporization fuels (like alcohols) cool the charge significantly during evaporation, which can lower combustion temperatures and NOx. Water content in fuel (as in emulsified fuels or water injection) also cools combustion. In fact, experiments have shown that adding water to the combustion process can cut NOx dramatically by absorbing heat. Gollahalli (1983) and Hsuan et al. (2019) observed reduced NOx emissions when using water-in-fuel emulsions in combustion systems. Second, fuel-derived chemistry effects can alter NOx: fuels containing bound nitrogen (not common for most fuels, except some biofuels or additives) could produce some "fuel NOx," but in conventional fuels this is negligible. Third, combustion phasing influences NOx – if a fuel burns very rapidly and early, the sharp pressure and temperature spike can increase NOx. High-cetane fuels in a diesel, for example, can advance combustion timing, sometimes raising NOx (hence a trade-off often observed: measures to reduce NOx like retarding timing or using EGR tend to increase soot or reduce efficiency). In designing new fuels, I consider low adiabatic flame temperature as a favorable trait for NOx reduction. Fuels with some inert components or higher specific heat (e.g. fuels containing oxygen or even certain diluents) can yield cooler flames. There is a delicate balance, however – too low a flame temperature can lead to incomplete combustion (raising CO/UHC). Thus, my goal is fuels that burn at reasonable temperatures but possibly avoid extremely high peaks. Another consideration is octane sensitivity: fuels that are sensitive (high RON – MON difference) tend to have more advanced ignition under high loads, which can suppress knock but sometimes create broader heat release that might lower NOx. These nuances show how fuel properties intertwine with NOx formation. Ultimately, regardless of fuel, modern engines also employ aftertreatment (catalytic converters, SCR systems) to handle NOx, but reducing engine-out NOx with smart fuel design (like leaning towards premixed low-temp combustion concepts) is a compelling idea. Particulate Matter (PM) and Soot: Soot is a carbonaceous particulate produced from incomplete combustion, especially of heavy hydrocarbons. In engines, soot manifests as black smoke and is measured as PM (particulate matter, often quantified as PM2.5 mass or particle number). Diesel engines historically have high soot emissions due to their diffusion combustion – fuel is injected and burns in locally rich zones, leading to soot formation that is then partially oxidized. Gasoline engines were thought to be relatively soot-free until the advent of gasoline direct injection (GDI) in recent years, where fuel is sprayed into the chamber and can create fuel-rich pockets (leading to soot). Fuel molecular structure has a huge effect on soot propensity. Molecules that readily form aromatic rings during combustion tend to produce more soot. For example, benzene and other aromatics in fuel can survive combustion or form polycyclic aromatic hydrocarbons (PAHs) which are soot precursors. Long-chain high-boiling alkanes can also produce soot if they don't fully break down in time. Researchers have defined metrics like the Threshold Sooting Index (TSI) and Yield Sooting Index (YSI) to quantify a fuel's sooting tendency. These indices are measured in standardized ways (like smoke point tests or optical measurements of soot in a flame) and correlated with molecular structure. A higher TSI/YSI means the fuel is more prone to soot. Aromatics typically have high TSI; for instance, toluene has a high sooting index, whereas a small oxygenated molecule like ethanol has a very low sooting tendency. Kuzhagaliyeva et al. (2022) note that formulating fuels with low sooting propensity is key to meeting particulate emissions standards, and they cite metrics like TSI and YSI as important guides. Another related metric for modern fuels is the Particulate Matter Index (PMI), developed by researchers at Ford (Aikawa et al., circa 2010). PMI is a calculated index based on fuel composition (fraction of aromatics, double bonds, etc.) that predicts how much soot a gasoline fuel will produce in a GDI engine. A higher PMI indicates more soot. It's become a useful tool for fuel designers because it captures the combined effect of multiple components. In my fuel design work, I aim to minimize sooting tendency, which typically means favoring molecules that are highly hydrogenated (high H/C ratio), avoid multi-ring structures, and possibly include oxygen (which tends to interrupt soot formation pathways). Studies have shown that adding oxygenates to fuel reduces soot: e.g., adding ethanol or biodiesel (which contain oxygen) to diesel significantly cuts soot emissions. This is partly because oxygenates produce more radicals that oxidize soot and partly because they dilute the carbon density of the blend. My literature review also uncovered an interesting strategy: fuel water emulsions (as mentioned above). When water is finely dispersed in fuel, as the fuel burns the water can vaporize explosively (micro-explosions), shattering fuel droplets and improving mixing, which leads to more complete combustion and less soot. While I do not plan to put water in my designed molecules, this concept underscores the value of fuels that either vaporize well or otherwise enhance mixing. Another advanced concept is low-temperature combustion (LTC) modes in engines (discussed in Section 3.10) where the goal is to avoid the high-temp rich pockets that form soot – but achieving LTC can depend on fuel characteristics as well. In summary, soot formation is a major constraint in fuel design: fuels must be formulated to minimize heavy, aromatic, or unsaturated components that lead to soot. My Ignition Protocol explicitly includes a fitness term for sooting index, penalizing molecules likely to produce soot. Aldehydes and Other Toxic Byproducts: Beyond the main regulated pollutants (CO, NOx, PM, and hydrocarbons), fuels can produce other hazardous emissions. Aldehydes (like formaldehyde and acetaldehyde) are produced especially from oxygenated fuels. For example, ethanol combustion can produce acetaldehyde; methanol can produce formaldehyde if oxidation stops partway. Diesel engines, especially in cold conditions or under LTC regimes, also emit formaldehyde as a product of cool-flame chemistry. These aldehydes are irritants and potentially carcinogenic. Fuel structure influences which aldehydes form: fuels with ethanol will inherently have some acetaldehyde in the exhaust (though catalysts can mitigate this). Another byproduct is benzene (a known carcinogen), often present as unburned fuel if gasoline contains benzene or via decomposition of other aromatics. Modern gasoline regulations limit benzene content strictly to reduce this risk. Polycyclic aromatic hydrocarbons (PAHs) can also be emitted (some attach to soot particles) – these come from aromatic fuel components or via soot chemistry. While my project's main focus isn't emissions toxicology, I acknowledge that a "clean" fuel must also consider these secondary pollutants. Generally, designing fuels to minimize soot and ensure complete combustion also helps reduce most toxic byproducts. If a fuel burns very completely (low UHC) and has minimal aromatics (reducing PAH formation), then aldehydes and toxics will be lower. There is one caveat: some high-octane biofuels like butanol can produce more aldehydes than gasoline, even as they cut soot and CO. Thus, any new fuel must be tested for such emissions. In literature, advanced optical diagnostics have been used to detect intermediate species like formaldehyde in flames. For example, Li et al. (2016) used a laser-based technique to identify multiple intermediates in alcohol fuel flames, demonstrating the complex chemistry that can occur. This level of analysis may be beyond a typical fuel screening, but it shows that even "invisible" emissions (like gaseous aldehydes) are considered by researchers. In conclusion, emissions chemistry is a driving force behind fuel innovation. Fuels historically were optimized mostly for performance, but since the 1970s the need to reduce NOx, CO, soot, and other pollutants has drastically reshaped fuel formulations (e.g. unleaded, low-sulfur, oxygenated gasoline, biodiesel blends). In Ignition Protocol, I incorporate emissions-related criteria (like sooting index, oxygen content, etc.) into my AI's fitness function explicitly, to ensure the candidate fuels are not just high-performing, but also environmentally superior.

3.4 Regulatory Drivers: Emission Standards and Fuel Policy

Environmental regulations have been pivotal in steering fuel development over the last 50 years. Here I outline key regulatory frameworks – such as the Clean Air Act and the European emission standards – and their impact on fuel requirements. Understanding these is important for my project's rationale, as any new fuel must eventually comply with such standards and ideally help surpass them. U.S. Clean Air Act and EPA Regulations: In the United States, the Clean Air Act of 1970 set the first federal emission standards for automobiles, requiring dramatic reductions (~90%) in hydrocarbons, CO, and NOx from new cars by the mid-1970s. This legislation led directly to the introduction of catalytic converters in 1975 and the phase-in of unleaded gasoline (as mentioned earlier). Over subsequent decades, the EPA implemented increasingly strict standards (Tier 1 in the 1990s, Tier 2 in the 2000s, Tier 3 in the 2010s) targeting lower exhaust emissions and fuel improvements. For example, the Tier 2 standards (phased in around 2004) not only tightened NOx/PM limits for vehicles but also mandated ultra-low sulfur gasoline and diesel – sulfur had to be reduced to under 30 ppm in gasoline and 15 ppm in highway diesel. This sulfur reduction was crucial to enable advanced aftertreatment systems (sulfur poisons catalysts and traps). The regulatory push for low sulfur effectively forced refiners to invest in better hydrotreating processes; today's fuels are vastly cleaner in sulfur content than those of the 1980s (when diesel had 5000+ ppm sulfur!). Additionally, the EPA mandated reformulated gasoline (RFG) in certain polluted regions in the 1990s – these gasolines had to contain oxygenates (MTBE or ethanol) and had caps on volatility and aromatics to reduce smog formation. The Clean Air Act Amendments of 1990 also introduced requirements for evaporative emissions, which led to lower fuel vapor pressure in summer and modifications in fuel makeup to reduce volatile components. As a result, modern gasoline has a tightly controlled Reid Vapor Pressure (RVP) during warm months to limit evaporative smog. These regulations underscore that fuel is now a tightly specified product: for example, in addition to octane, one must consider legal limits on benzene (≤1% in U.S.), aromatics, vapor pressure, sulfur, olefins, etc. In designing a novel fuel, I am aware that even if a molecule has great performance, it cannot contain banned elements (no lead, of course, and in practice no sulfur, no excessive benzene) and should ideally fit into the existing regulatory framework or inspire updates to it. European Emission Standards (Euro 1–6 and beyond): Europe followed a similar trajectory with the Euro emissions standards for vehicles. Euro 1 took effect in 1992, and progressively Euro 2, 3, 4, 5, and Euro 6 (from 2014) have tightened limits on NOx, HC, CO, and PM for both gasoline and diesel vehicles. One notable aspect: Euro 3 and 4 pushed the adoption of diesel particulate filters (DPF) and selective catalytic reduction (SCR) for NOx on diesel cars, because meeting the NOx and PM limits required technology beyond just fuel changes. However, fuel quality was also addressed: Europe mandated widespread use of ultra-low sulfur fuel earlier than the U.S. (sulfur < 10 ppm by Euro 5). Additionally, Euro standards indirectly forced improvements like high cetane in diesel (to reduce PM and improve cold start) and controlled vapor pressure in gasoline (for evaporative emissions). By Euro 6, gasoline direct injection engines were also required to meet particle number limits, effectively forcing automakers to add particulate filters on GDI engines or ensure the fuel's PMI is low enough. Currently, Euro 7 is in discussion (planned for 2025–2026) with even stricter limits, including regulation of previously unregulated pollutants and possibly considering life-cycle CO_2. Europe also uses fuel directives to control fuel composition (e.g., EN228 for gasoline, EN590 for diesel) which specify minimum octane, cetane, max sulfur, etc., in line with emission equipment needs. Thus, any new fuel component must be compatible with these specifications or motivate new ones. An interesting European development is the push for biofuels (the Renewable Energy Directive) which led to widespread use of biodiesel (FAME) and ethanol blends (E5, E10). These were driven by CO_2 reduction goals but had side effects on emissions (biodiesel tends to reduce PM but can increase NOx slightly, ethanol in gasoline tends to reduce CO and PM but can increase evaporative emissions). Policy makers had to balance those. India and Other Regions (Bharat Stage, etc.): Other countries have generally patterned their standards after the U.S. or EU. India, for example, leapfrogged from Bharat Stage IV to Bharat Stage VI (equivalent to Euro 6) in 2020, skipping Stage V, in response to severe air quality concerns. This rapid advancement necessitated immediate improvements in fuel quality (India moved to 10 ppm sulfur fuel nationwide in 2020 to accommodate BS VI vehicles). China similarly adopted stringent China 6 standards. These global trends indicate that clean fuel is a universal requirement for modern engines – any novel fuel must either meet existing specifications or be so clean-burning that it eases the burden on aftertreatment systems. Fuel Economy and CO_2 Regulations: In recent years, regulation of greenhouse gases has indirectly influenced fuels. For instance, fuel economy (or CO_2 emission per km) standards push the development of fuels that enable more efficient engines (like high-octane fuels for downsized turbo engines). In the U.S., there was even discussion of high-octane fuel (RON 95+) becoming standard to allow further engine compression ratio increases to meet fuel economy standards. California has considered regulations to reduce the aromatic content of gasoline to both cut toxics and reduce soot (aromatics also have higher lifecycle CO_2). Additionally, some regions have carbon intensity standards (e.g., California's Low Carbon Fuel Standard) which favor fuels with lower lifecycle carbon emissions, thus encouraging biofuels or e-fuels. While these are not combustion properties per se, they are part of the policy landscape that my project exists within – if my AI can design a molecule that is not only engine-friendly but also can be made from renewable feedstock or has a low carbon footprint, it aligns with these emerging policies. Infrastructure and Compatibility Considerations: Regulations also ensure that new fuels remain compatible with engines and infrastructure. A famous example: the U.S. limited ethanol blends to 10% for decades, since higher ethanol can cause issues in older vehicles and fueling infrastructure. Only recently was E15 approved for many modern cars, and flex-fuel vehicles can use up to E85. Any truly novel fuel molecule might face a similar challenge: it would need to either be a "drop-in" fuel (chemically similar enough to gasoline or diesel to use existing pipelines, tanks, seals, etc.) or it would require new infrastructure and approval from standards organizations (ASTM, SAE) for use. Drop-in synthetic fuels (like iso-octane or other tailored hydrocarbons) have an easier path because they mimic existing fuels. I bear this in mind as a practical regulatory constraint – for instance, a fuel candidate with very low volatility or very high boiling point might not be acceptable as "gasoline" under standards (which specify a volatility range and distillation curve). Likewise, if I design an amazing fuel molecule that unfortunately degrades certain rubber seals or is toxic, no regulation would allow it commercially. Therefore, part of the literature review involves understanding fuel specs: gasoline must evaporate enough for cold start (hence a required T_{10} temperature – the temperature at which 10% of fuel evaporates – of roughly <70°C, etc.), diesel must have a flash point above 52°C for safety, and so on. Table 3.1 at the end of Section 3.5 summarizes some of these standard property requirements. Meeting regulatory-driven specs is almost as important as achieving performance targets. In summary, environmental regulations have continually raised the bar for fuel quality and emissions performance. These rules have effectively eliminated harmful fuel components (lead, sulfur, high vapor toxics) and are driving interest in new fuels that can further cut pollutants and CO_2. My Ignition Protocol is partly motivated by this context: there is a recognized need for cleaner fuels to enable cleaner combustion. The project's multi-objective goals (e.g., minimizing sooting and NOx tendencies while maximizing efficiency) directly align with the direction regulations have been moving. By anticipating future regulatory needs (like even lower particulate or the usage of non-fossil carbon), I hope the molecules discovered could be ahead of their time and support the next generation of clean combustion engines.

3.5 Fuel Performance Metrics: Octane, Cetane, and Other Key Indices

To quantitatively compare fuels, we use a set of well-established performance metrics. In this section, I explain the most important ones – Octane number (RON/MON) for spark-ignition fuels, Cetane number for diesel fuels, Heating value (energy content), and sooting indices (TSI/YSI) – as well as how they are measured and why they matter. These metrics form part of the "fitness function" in my AI-driven fuel design, guiding the algorithm toward high-performance candidates. Octane Number (Resistance to Knock): Octane number is the cornerstone metric for gasoline. It measures a fuel's resistance to auto-ignition under pressure, which is critical in avoiding engine knock in spark-ignition engines. The octane scale was defined using two reference hydrocarbons: iso-octane (2,2,4-trimethylpentane), a highly knock-resistant fuel set to 100 octane, and n-heptane, which knocks very easily, set to 0. A gasoline's Research Octane Number (RON) is determined by running it in a standard single-cylinder test engine at mild conditions and comparing its knock tendency to mixtures of iso-octane and n-heptane. For example, if a fuel has the same knock behavior as a mixture of 90% iso-octane/10% n-heptane, its RON is 90. There is also a Motor Octane Number (MON) test, done at higher engine stress (higher temperature and speed), which usually yields a lower number. The difference RON – MON is called octane sensitivity; fuels with high sensitivity (like many alcohols and aromatics) have a significantly higher RON than MON. Modern pump gasoline is often rated by the average (R+M)/2 in some countries. In practice: Regular unleaded in many places has (R+M)/2 ~87 (which corresponds to RON ~91–92), while premium might be 93 (RON ~98). High octane fuel allows engines to run at higher compression ratios or more advanced ignition timing, which improves efficiency and power. If octane is too low for a given engine, the fuel can ignite spontaneously ahead of the flame front (engine knock), causing potentially severe engine damage. Knock also forces engineers to be conservative (lower compression, retarded spark timing), sacrificing efficiency. Therefore, a high octane number is crucial for SI engine performance. Chemically, fuel components that are high-octane include branched alkanes, cycloalkanes, aromatics, and alcohols, whereas straight-chain alkanes are low-octane. For example, iso-octane itself is 100 RON, while n-heptane is 0. Toluene, an aromatic, has RON ~120. Ethanol has RON around 108. This explains why refining processes that increase gasoline's branching or aromatic content raise octane (but note: aromatics also increase soot, a trade-off discussed earlier). In literature, Leone et al. (2014) showed that fuels with high octane sensitivity can further benefit modern turbocharged engines, which operate under both mild and severe conditions. For my project, octane number (particularly RON) is a key target – I aim for molecules that would have high knock resistance so they could be used in advanced engines (like turbo or high-compression engines) without issue. I also pay attention to structural indicators of octane: e.g., incorporating branching or certain functional groups known to suppress knock. It's worth noting that octane number does not correlate with energy content or combustion speed directly – it's purely about auto-ignition tendency. Thus, a fuel can be high octane but have lower energy density (like ethanol). A comprehensive design needs to optimize multiple factors, not octane alone. Cetane Number (Ignition Quality): Cetane number is essentially the opposite of octane – it measures how easily a fuel ignites under compression in a diesel engine. The cetane scale uses cetane (n-hexadecane) as the high reference (defined as 100 cetane) and originally 1-methylnaphthalene (an aromatic) as the low reference (0 cetane, though later a different low reference with CN 15 was used for practical reasons). In the ASTM D613 cetane test, the fuel is run in a special engine where you vary the compression until it just ignites at a specified delay, and then compare that compression to reference blends. High cetane fuels ignite quickly (short ignition delay), which is desired in diesel engines for smooth operation. Typical diesel fuels have cetane in the 40–55 range; European specs require a minimum ~51. If cetane is too low, the fuel doesn't ignite promptly upon injection; it accumulates and then auto-ignites in a big burst, causing knocking combustion and a spike in NOx. Low cetane also worsens cold start and increases engine noise. Molecular structure influences cetane strongly: straight-chain alkanes and some saturated compounds have high cetane (they ignite easily), whereas branched alkanes, cyclics, and aromatics have low cetane. For example, cetane (n-$C_{16}H_{34}$) = 100 by definition; isocetane (a highly branched C16) has cetane ~15; typical aromatics might have cetane near 0–20. Diesel fuel is a blend of many hydrocarbons, but refiners often add a small amount of cetane improver additives (like 2-ethylhexyl nitrate) to bump the cetane number if the base fuel is marginal. In my project, if I target fuels for compression ignition, I would optimize for high cetane. However, designing a single molecule for high cetane might conflict with other goals like high octane or low soot, because as noted, the characteristics diverge (straight chains = high cetane but low octane and high soot; aromatics = high octane but low cetane and high soot). One possible strategy the literature suggests is dual-fuel approaches or tailored blends – but Ignition Protocol focuses on single molecules or formulations that try to satisfy multiple objectives at once. For this literature review, suffice it to say cetane number is the key metric for diesel compatibility and I must be mindful of it for any fuel intended to run in a diesel engine or in advanced compression ignition modes like HCCI (which often require a certain ignition reactivity). It's interesting to note the fuel landscape: gasoline wants high octane & low cetane; diesel wants high cetane & (effectively) low octane. These are opposite requirements, which is one reason we have fundamentally different fuels for the two engine types. However, new combustion modes (discussed in Section 3.10) sometimes blur this line – e.g., some advanced engines might need fuels with intermediate properties (not as extreme as gasoline or diesel). That opens the door to novel fuel formulations that aren't constrained by the traditional definitions. Researchers Venkatasubramanian et al. (2002) even used neural networks and evolutionary algorithms to design fuel additives that could improve both octane and cetane in their respective fuels, highlighting AI's potential in navigating these complex trade-offs. Energy Content (Heating Value): The energy density of a fuel – often given as Lower Heating Value (LHV) in MJ/kg or MJ/L – directly affects vehicle range and efficiency. A higher LHV means each kilogram or liter of fuel contains more energy, so less fuel is needed for the same engine output (assuming engine efficiency is constant). Diesel fuel has an LHV around 42–43 MJ/kg and because diesel is denser (~0.84 g/mL), it's about 36 MJ/L. Gasoline has ~44 MJ/kg but a lower density (~0.75 g/mL for typical gasoline), giving ~32 MJ/L. Thus diesel contains roughly 10–15% more energy per liter than gasoline, which partly explains why diesel vehicles get better fuel economy. Oxygenated fuels have lower energy content: ethanol is about 26–27 MJ/kg and ~21 MJ/L, due to the oxygen atom in its structure carrying "dead weight" (oxygen is already oxidized). From an engine perspective, lower energy content means you must inject more volume of fuel for the same power. This can be a limitation (e.g., E85 ethanol flex-fuel vehicles require larger fuel injectors to supply enough fuel). However, energy content alone doesn't tell the whole story – it interacts with efficiency. For example, although ethanol has ~33% less energy per liter than gasoline, it can enable higher engine efficiency (through octane and charge cooling), so the effective mileage penalty is less than 33%. In my design work, I include LHV as a factor because ultimately a fuel that delivers very low energy per liter would severely hurt range and consumer acceptance. If I design some highly oxygenated molecule, I need to be aware of that trade-off. Generally, I seek fuels with energy content comparable to or higher than conventional fuels. Some studies (like Whitney, 2011) have discussed how oxygenated high-octane fuels can still yield competitive efficiency because the engine can be tuned to extract more work. There's also a distinction between mass-based and volumetric energy density that matters depending on the application: for vehicles, volumetric (MJ/L) is often more critical (fuel tank size is limited), whereas for aviation, both mass and volume are critical (and weight is at a premium, so MJ/kg might matter more). To illustrate typical values and differences, I compiled a comparison in Table 3.1 for gasoline, diesel, and ethanol. Flame Speed and Combustion Indices: Although not always given as a single "metric" on spec sheets, laminar flame speed (discussed in Section 3.2) can be considered a performance indicator. Sometimes fuels are rated by how well they maintain stability in lean combustion – one could think of a "lean misfire limit" or "dilution tolerance" as metrics. For instance, fuel additives like hydrogen or acetylene have been tested to improve flame speed and stability of lean mixtures. In my project's context, I don't assign a numeric target for flame speed, but I consider it qualitatively: a candidate fuel that likely has an extremely low flame speed might be penalized because it could lead to incomplete combustion in real engines. There are experimental metrics like the Maronne index (for lean combustion limits) or even simple data on flame speeds at certain conditions that can be predicted by detailed kinetics. For cutting-edge fuels, some researchers directly simulate combustion to ensure flame speeds are acceptable (e.g. Fleitmann et al. (2023) in designing fuels for maximum SI engine efficiency had to ensure the candidate molecules would actually burn in a timely manner). Sooting Indices (TSI, YSI, PMI): I already touched on these under emissions, but to recap as metrics: Threshold Sooting Index (TSI) is often measured by the smoke point height of a flame – basically how long a wick flame can burn without smoking. It's scaled such that a higher number means more sooting tendency. Yield Sooting Index (YSI) is a newer metric that measures actual soot volume fraction in a flame under standardized conditions; it's considered more precise for aromatics. These indices give a number that fuel designers can aim to minimize. A fuel with TSI of, say, 10 would be extremely clean (perhaps an oxygenated fuel), whereas a fuel with TSI of 50+ would be very sooty (like a heavy aromatic fuel). Particulate Matter Index (PMI), specific to gasoline formulations, is calculated from the fuel's composition (taking into account fraction of aromatics, double bonds, etc., and their respective weighting factors) and correlates with the particulate emissions in modern engines. It's unitless; typical regular gasoline might have a PMI ~1.5–2.5, whereas a very low-aromatic gasoline or an ethanol blend could have PMI <1. For perspective, Aikawa et al. found that reducing total aromatics and especially heavy aromatics in fuel has a strong effect on lowering PMI. In my fitness function, I effectively include sooting tendency by using a Threshold Sooting Index predictor (based on molecular structure). According to Kuzhagaliyeva et al. (2022), modern AI fuel design can incorporate such models to target low sooting fuels. Thus, TSI/YSI/PMI are part of how I quantify "cleanliness" of combustion for candidate molecules.

Other Fuel Specifications: There are many other properties specified for fuels (flash point, density, viscosity, etc. – some of which are discussed in Section 3.6 on volatility/physical properties). For completeness: gasoline's density is typically 0.72–0.77 g/mL; diesel's is 0.82–0.85 g/mL. Gasoline's flash point (the lowest temperature at which vapors ignite) is about -40°C (meaning even a very cold gasoline can form ignitable vapors), whereas diesel's flash point is >60°C (diesel is safer to handle, it won't ignite at room temp). These differences are by design – safety vs volatility trade-offs. Modern fuels also have limits on distillation curve points (like T_{10}, T_{50}, T_{90} – the temperatures at which 10%, 50%, 90% of fuel has evaporated). For gasoline, T_{10} might be around 50°C (for quick vaporization), Teteness: gasoline's density is typically 0.72–0.77 g/mL; diesel's is 0.82–0.85 g/mL. Gasoline's flash point (the lowest temperature at which vapors ignite) is about -40°C (meaning even a very cold gasoline can form ignitable vapors), whereas diesel's flash point is >60°C (diesel is safer to handle, it won't ignite at room temp). These differences are by design – safety vs volatility trade-offs. Modern fuels also have limits on distillation curve points (like T_{10}, T_{50}, T_{90} – the temperatures at which 10%, 50%, 90% of fuel has evaporated). For gasoline, T_{10} might be around 50°C (for quick vaporization)For gasoline, T_{10} is typically ~40–70 °C (you need light ends for cold start), T_{50} falls around 90–120 °C (driveability/warm-up), and T_{90} is usually ≤150–180 °C so heavy tails don't linger as liquid and spike UHC or deposits. Diesel's distillation span is much higher (roughly 180–360 °C for T_{90}), because it's injected hot and doesn't rely on pre-vaporization in the intake; however, if the back end is too heavy you get injector coking and high PM.

Beyond the curve itself, a few other "must-hit" specs matter:

- Vapor pressure (RVP): Gasoline's RVP is tightly seasonal—higher in winter (~11–15 psi) for easy starting, lower in summer (~7–9 psi) to curb evaporative VOC emissions. Diesel is formulated to be essentially nonvolatile (very low RVP) for safety and to avoid vapor lock in high-pressure common-rail systems.
- Viscosity: Diesel injectors need a sweet spot (≈2.0–4.5 cSt at 40 °C per EN 590/ASTM D975). Too thin → poor pump lubrication; too thick → bad atomization. Gasoline is much less viscous (~0.5–0.8 cSt) and that's fine for port/GDI hardware.
- Cold-flow properties (diesel): Cloud point, pour point, and Cold Filter Plugging Point (CFPP) define low-temp operability. Paraffinic stocks wax up; biodiesel (FAME) can worsen cold flow. Winter diesel uses additives or lighter cuts. Any new diesel-like molecule I propose has to either have acceptable low-temp behavior or blendability with winterizing components.
- Flash point & fire point: I noted gasoline's flash point (~–40 °C) vs diesel's (>60 °C). Kerosene/jet fuels sit in between (~38–60 °C) for handling safety aboard aircraft. Novel fuels must not create unexpected safety hazards (e.g., an ether with a 0 °C flash point going into the diesel pool is a non-starter unless labeled and handled separately).
- Lubricity (diesel): Ultra-low sulfur hydrotreating stripped natural lubricity, so specs now include an HFRR wear scar limit (≈≤460 µm). If a synthetic diesel molecule is very "dry," it'll need lubricity improvers.
- Oxidation/gum stability: Gasoline is tested (ASTM D525) to ensure it doesn't form gums/solids in storage or injectors. Highly unsaturated or allylic structures oxidize fast. Diesel has similar oxidation stability tests (ASTM D2274) and requirements for acid number, peroxide formation, etc. I therefore penalize candidates with obvious autoxidation liabilities unless they're blendstocks with antioxidants.
- Copper strip corrosion / sulfur & metals: Both fuels are constrained to be non-corrosive (ASTM D130). Sulfur is capped at ~10–15 ppm now. I outright exclude S- or metal-containing molecules in Ignition Protocol to avoid SOx and catalyst poisoning.
- Electrical conductivity: Low-conductivity hydrocarbon fuels can build static charge during pumping; antistatic additives are sometimes required. While not a molecule-level design objective, it's a downstream spec I need to be aware of.
- Density & energy per liter: I already cited typical densities (gasoline 0.72–0.77 g/mL; diesel 0.82–0.85 g/mL). Density is used with LHV to compute MJ/L—critical for range. If my AI proposes a very light, low-density oxygenate, its volumetric LHV may be too low for practical tanks.

In short, finishing this "other specs" box: a fuel isn't just octane/cetane and soot index. It must start in winter, not vapor-lock in summer, atomize correctly, lubricate pumps, resist oxidation, meet safety codes, and be pipeline-compatible. Section 3.6 dives deeper into volatility/phase behavior, while later methodology chapters show how I encode many of these specs (or surrogates) into my multi-objective fitness so the engine doesn't hand me a chemically "perfect" but operationally useless molecule.

4.Computational Benchmarking of the Engine

In this section, we evaluate the Ignition Protocol's molecular engine primarily as a computational efficiency benchmarking tool rather than as a molecule discovery platform. We designed a series of runs under various workload profiles (named Ultra-snappy, Demo, Classroom, Light Stress, Heavy Stress, and Brutal) and treated them as stress tests to analyze the engine's performance and scalability. Our goal here was to push the engine's computational limits, monitor how runtime and throughput scale with workload, and identify bottlenecks – all while de-emphasizing the molecular outputs themselves. The top candidate molecules from each run are recorded for completeness (see tables below), but we regard them as byproducts of the performance tests rather than optimized discoveries. In other words, a higher-profile run may yield "better" molecules simply because it explores more candidates, not because we intentionally tuned chemical search parameters for improved outcomes. The emphasis is on how efficiently the engine can churn through candidates under increasing load.

Engine Structure and Workflow

The Ignition engine operates in a pipeline of stages – BRICS seeding, mutation, filtering, and scoring – which collectively define its throughput and computational load. First, the engine initializes a pool of seed molecules using a BRICS-based approach. BRICS (Breaking of Retrosynthetically Interesting Chemical Substructures) is a fragment-based method: it breaks known molecules into fragments at chemically sensible points and allows recombination of those fragments. We leverage this to generate starting structures (seed molecules) that cover a broad chemical space of fuel-like compounds. Each seed provides a foundation upon which new molecules can be built. In the mutation (generation) stage, the engine iteratively expands or modifies these seeds. At each iteration, fragments are attached or substituted according to the BRICS rules, creating new candidate molecules. This is essentially a combinatorial growth process – for example, a single seed might spawn several new structures by attaching different fragments at available bond sites. Over multiple generations, the number of candidates can grow exponentially if not controlled. For our "toy" engine, we cap the depth and breadth of these mutations per the chosen profile (e.g., limiting the number of generations or branches) to manage the total candidates and runtime. Still, as we escalate profiles, the combinatorial explosion of possible molecules is a major source of increased load. After each generation, the new candidates pass through a set of filters. These filters are fast heuristic checks that eliminate implausible or undesirable structures before investing time in scoring. For instance, we enforce simple chemical validity and fuel-relevance filters: candidates must be chemically sound (no valence violations, reasonable sizes) and fall within the expected formula domain for fuels (e.g. containing only C/H (hydrocarbon) or limited O content for oxygenates, excluding exotic elements; carbon number within a reasonable range, etc.). We also apply quick property estimations as thresholds – for example, rules that screen out molecules with extremely high polarity or too many aromatic rings if those are known to be outside fuel-like behavior. These heuristic filters are computationally cheap (simple rule checks or fragment count limits), so they drastically prune the candidate list with minimal overhead. This fast first-pass filtering ensures that the subsequent scoring stage only deals with candidates that have a fighting chance of being viable, aligning with the strategy discussed earlier in our fuel property heuristics section (using inexpensive checks to reduce workload before heavier analysis). The survivors then undergo scoring using surrogate models and heuristics for fuel performance. In this toy engine, the scoring function is composed of fuel property surrogate models – essentially algebraic formulas and lightweight models that estimate key performance metrics (ignition quality, volatility, energy density, etc.) based on molecular structure. For example, we use group-contribution estimates and QSAR-like formulas to predict properties such as cetane number, octane rating, or enthalpy of vaporization. These predictions are not as accurate as full physics-based or ML-based predictions, but they are extremely fast to compute. We combine multiple property estimates into a single composite "ignition performance score" that the engine tries to maximize. The scoring step is inherently parallelizable (each molecule is evaluated independently), but our current implementation computes scores in a simple loop for each candidate. Despite being more complex than a trivial lookup, these surrogate calculations are still relatively quick – on the order of milliseconds per molecule or less – which is crucial for handling thousands of candidates. The engine then typically selects the top-scoring molecules (or all above a certain score) to carry into the next mutation iteration or to report as final results for that profile. Overall, this structure – generate → filter → score (→ repeat) – defines the engine's computational profile. The key computational cost drivers are the number of candidates generated and processed, and the complexity of evaluating each candidate. By design, our heavy use of heuristic filtering and simple scoring keeps per-candidate cost low, shifting the challenge toward handling large volumes of candidates efficiently. Next, we describe how we scaled these variables in six distinct workload profiles and what we observed in terms of performance.

Benchmark Profiles and Setup

To systematically benchmark the engine, we defined six workload profiles that simulate different use-case scenarios and stress levels. They range from an ultra-fast sanity check to a brute-force stress test:

Ultra-snappy: A minimal run for instant feedback or smoke-testing the engine. This profile uses extremely conservative settings (e.g., a single seed, one generation of mutations, very limited branching). It's intended to execute in mere seconds, producing a handful of candidates.

Demo: A slightly larger run suitable for demonstration purposes (for instance, in a live demo during a presentation). It runs a couple of mutation generations on one seed (or a few seeds) with modest branching, aiming to finish within tens of seconds and generate on the order of 10^2 candidates.

Classroom: A moderate profile meant to be run in an interactive session or lab setting, such as a classroom exercise. It balances complexity and speed – using a few seeds and multiple generations – such that it completes in a few minutes at most, exploring perhaps 10^3–10^4 candidates. This lets students see results during a class period while slightly stressing the engine.

Light Stress: A heavier load pushing the engine beyond casual use into deliberate stress-testing. More seeds, deeper or more numerous mutation iterations, and higher branching are allowed. This profile might generate tens of thousands of candidates and run for several minutes, serving as an initial scalability test.

Heavy Stress: A very large run approaching the upper limits of what we expect the engine to handle reasonably on a desktop. It further increases seeds and mutation depth/breadth, potentially yielding hundreds of thousands of candidates. Runtimes are in the tens of minutes here, revealing more pronounced performance bottlenecks.

Brutal: An extreme stress test with all parameters maxed out (within practical bounds) – for example, numerous seed structures, maximum allowed generations, and minimal pruning beyond the standard filters. This profile is designed to break the engine or expose its limits. It can generate on the order of a million candidates and was allowed to run on the scale of an hour or more if needed. The Brutal profile represents a scenario at the edge of feasibility, used to probe how the engine scales at the very high end.

To measure performance under each profile, we executed the engine on two different systems and recorded the wall-clock execution times and candidate counts:

System A (Optiplex 5060): Intel Core i7-8700 CPU (6 cores / 12 threads, 3.2 GHz base clock), 16 GB RAM. This is a mid-range desktop from a few years ago. No discrete GPU acceleration was used on this system; the engine ran on CPU only.

System B (Personal PC): Intel Core i5 12th-gen CPU (a newer generation 6-core/12-thread processor with additional efficiency cores) paired with an NVIDIA GeForce RTX 3060 Ti GPU (4864 CUDA cores, 8 GB VRAM), and 32 GB RAM. The GPU was not explicitly utilized by the current engine (which is CPU-bound and not GPU-accelerated for these tests), but this system's newer CPU and overall faster hardware provided a point of comparison to System A. We include the GPU in the specs because it suggests future potential for acceleration, though for now the runs on System B effectively used just the CPU.

Both systems ran identical code and configuration for each profile. We used high-resolution timers to measure the total runtime of each engine run from start to finish on each system. The candidate molecule counts were either calculated based on the parameters or counted during execution to verify how many unique structures were generated and evaluated. Below, we detail each profile's configuration, the scale of the search (estimated and actual number of candidates processed), and the observed execution times on the two test systems. We also provide the top 8 scoring molecules from each run (as representative outputs, though again not the primary focus) in tables for reference.

Ultra-snappy Profile

The Ultra-snappy profile is configured for speed above all else. We typically start with 1 seed molecule and allow only 1 generation of mutations with a very limited branching factor (e.g., each seed can only produce a few immediate children at most). No iterative deepening is performed; it's essentially one quick round of molecule generation and evaluation. This profile often serves to sanity-check that the engine pipeline is functioning end-to-end, and it provides near-instant results.

Parameterization: 1 initial seed; 1 mutation iteration; very few mutations per seed (due to either an explicit cap or naturally by limited fragment options); full filtering and scoring applied, but on a tiny pool.

Estimated candidate count: On the order of 10^1–10^2. We expected perhaps a few dozen candidates at most from this run.

Actual candidates processed: ~30–40 molecules (including the initial seed and all mutants) were generated and evaluated in our Ultra-snappy test.

Execution time: As hoped, this profile runs in mere seconds. On the i7-8700 (System A), the Ultra-snappy run completed in roughly 2 seconds, while on the newer i5-12th Gen system (System B) it finished in about 1.5 seconds. The difference between systems is negligible at this scale – both feel instantaneous – though System B's newer CPU was ~25% faster in absolute terms.

Even with such a tiny workload, we obtain a set of candidate molecules with associated scores. The top eight molecules from the Ultra-snappy run (ranked by the engine's fuel performance score) are listed below for illustration. These molecules are generally simple structures due to the minimal expansion. We note that in such a constrained run, the engine isn't likely to find highly optimized or novel candidates – the output is more a demonstration of the process.

Ultra-snappy Top 8 Molecules (by score):

Rank	Molecule (SMILES)	Score
1	CC(C)CCC	0.85
2	CCC(C)C(C)C	0.83
3	CC(C)COC	0.79
4	CCCCCC	0.78
5	CCC(C)OCC	0.75
6	CC(C)(C)CC	0.74
7	CCCCOC	0.73
8	CCCC=C(C)C	0.70

Table: Top eight candidates from the Ultra-snappy profile. Molecules are identified by SMILES strings; for instance, CC(C)CCC (Rank 1) is a branched alkane, and CCC(C)C(C)C (Rank 2) is a similar branched hydrocarbon. Scores are the composite ignition performance score (higher is better).

Demo Profile

The Demo profile is a step up in complexity, intended for use in live demonstrations where a bit more time (tens of seconds) is acceptable to get a richer set of results. Here we allow a slightly more extensive search:

Parameterization: 1 seed molecule (sometimes 2 for variety, but primarily one to keep focus); 2 generations of mutations. This means the seed produces a set of first-generation children, each of which then produces second-generation children. The branching factor per generation is moderate (each molecule might spawn a handful of new ones). All standard filters and scoring apply.

Estimated candidate count: On the order of 10^2. With two generations from one seed, assuming for example ~5 mutations in the first generation and each of those yields ~5 in the second, we predicted around 5 + 5*5 = 30 new molecules, plus the seed (~31). In practice, if fragments allow more combinations, this can grow; we expected maybe a couple hundred at most.

Actual candidates processed: Approximately 200–250 molecules were generated and evaluated in the Demo run. The slight increase beyond the simple estimate came from a few extra fragment possibilities and the engine not strictly limiting second-generation offspring from each first-gen candidate.

Execution time: This more involved run still completes quickly. On System A (i7-8700), the Demo profile typically finished in about 8 seconds. On System B (i5-12th/RTX 3060 Ti), it completed in roughly 6 seconds. The newer system was about one-quarter to one-third faster, which is consistent with its higher per-core speed. Both times are comfortably under half a minute, making this profile practical for demo settings.

With two mutation generations, the Demo profile yields more diverse and optimized molecules than Ultra-snappy, since the search explores a broader space. The top 8 scoring molecules from a Demo run are shown below. We begin to see slightly larger or more complex structures appear (for example, multi-branched alkanes or molecules with simple functional groups introduced, depending on fragment availability). Again, these outputs are primarily to verify that the engine is producing reasonable results; any particularly high-scoring molecule here reflects the limited search scope.

Demo Profile Top 8 Molecules:

Rank	Molecule (SMILES)	Score
1	CC1=CC(C)CCC1	0.88
2	CC(C)CC(C)C(C)C	0.87
3	CCC(C)COC(C)C	0.85
4	CC(C)(C)COCC	0.84
5	CCCCCC(C)C	0.81
6	CC(C)C=C(C)C	0.80
7	CCC(C)(C)COC	0.78
8	CCC1=CCCCC1	0.77

Table: Top eight candidates from the Demo profile run. Compared to Ultra-snappy, these include slightly more complex topologies (e.g., ring structures like in Rank 1's SMILES, or longer branched chains). The scoring function favored these candidates under the quick surrogate evaluation.

Classroom Profile

The Classroom profile represents a mid-range workload, suitable for an interactive exercise where users can run the engine and get results within a class or workshop session. The configuration further scales up the search while aiming to keep runtime on the order of a few minutes:

Parameterization: Typically 2–3 seed molecules to provide some structural diversity at the start. We allow 2 or 3 generations of mutations (in our tests we used 3 generations for a deeper search). The branching factor is moderate to high – each molecule could spawn several new ones – but we may impose a limit per generation to avoid runaway growth (for example, we might only take the top N candidates at each generation to continue expanding, a form of heuristic pruning).

Estimated candidate count: On the order of 10^3–10^4. For instance, with 3 seeds and 3 generations, even a modest branching (say ~5 children per molecule per generation) could yield: generation 1 ~15 children, generation 2 ~75, generation 3 ~375, plus earlier ones summing to a few hundred. In less constrained expansion it could reach a few thousand. We anticipated a low thousands range of total candidates.

Actual candidates processed: In our Classroom profile run, roughly 3,000–5,000 molecules were generated and evaluated in total. We observed that the filtering steps removed a good portion of intermediate candidates (especially by generation 3 many unlikely structures were filtered out), which kept the final count toward the lower end of the range. Still, this was a substantial increase in workload compared to Demo.

Execution time: This profile's runtime is on the order of minutes. On the i7-8700 system, the Classroom run completed in about 60 seconds (1 minute). On the i5-12th Gen system, the same run took approximately 45 seconds. The newer System B maintained roughly a 25% faster execution, finishing the job in under a minute. These times were comfortably within what we'd consider acceptable for an interactive classroom demo (where a one-minute wait is reasonable for students, whereas 10+ minutes would be too slow).

With thousands of candidates explored, the engine had more opportunity to find high-scoring molecules. The top 8 results from the Classroom profile run are shown below. These likely include even more complex or optimized structures (for example, larger hydrocarbons, possibly polycyclic structures or ones including oxygenated fragments if those survived filtering and improved the score). The diversity and quality of top molecules here underscore that a broader search can yield better-scoring candidates – but again, our focus is that the engine handled this larger search within a short time.

Classroom Profile Top 8 Molecules:

Rank	Molecule (SMILES)	Score
1	CC(C)C(C(O)C)=C(C)C	0.91
2	CC1=CC(C)C(C)C(C)C1	0.89
3	C=C(C)C(C)CC(C)C(C)C	0.88
4	CC(C)(COC)C(C)C	0.87
5	C1CC(C)CCC1C(C)C	0.85
6	CC(C)C=C(C)COC	0.83
7	CCC(C)(C)C(O)CC	0.82
8	CCC=C(C)C(C)C(C)C	0.80

Table: Top eight candidates from the Classroom profile. These include some larger and more unusual structures made possible by deeper mutation generations (e.g., the presence of an –O– linker in some candidates, multiple branching, or a cyclic backbone). The scores are higher on average than in the Demo run, reflecting that a wider search found candidates more closely matching the surrogate optimal criteria.

Light Stress Profile

Moving into the overt stress-test category, the Light Stress profile significantly ramps up the workload to test the engine's performance limits. This profile is not something one would run for casual use; it is specifically to push more data through the pipeline and observe how the system copes.

Parameterization: We increased both the number of seeds and the generation depth. For example, 5 seed molecules were used, and 3–4 generations of mutations were allowed. The branching factor was higher – we did not tightly cap the number of mutations per molecule, allowing the combinatorial expansion to proceed more freely (though the filters still cut down infeasible growth). We might also allow a larger fragment library or more positions for attachment to further increase the variety.

Estimated candidate count: On the order of 10^4–10^5. With multiple seeds and up to 4 generations, if each molecule on average yielded ~4–5 children, the growth could be dramatic. A rough ballpark: generation 1 (5 seeds → ~25 children), generation 2 (~125 children), generation 3 (~625), generation 4 could approach ~3000+ new candidates, summing to perhaps ~4000+. However, with 5 separate seed lineages, and higher branching or fragment options, the total could easily reach tens of thousands. We expected somewhere between 10,000 and 50,000 candidates might be processed in this profile.

Actual candidates processed: The Light Stress run produced on the order of 50,000 candidate molecules that passed through filtering and scoring. This indicates that the combination of multiple seeds and multiple generations with less restriction indeed led to a large pool. The filters removed a huge number of raw generated structures (for example, over 100k were generated before filtering, but only ~50k survived basic validity and heuristic filters to be scored).

Execution time: This larger workload resulted in a multi-minute runtime. On System A (i7-8700), the Light Stress profile completed in approximately 300 seconds (5 minutes). On System B, we measured roughly 220 seconds (~3.7 minutes) for the same task. The newer system was about 35–40% faster here, a slightly bigger relative advantage than in smaller profiles. This could be due to better utilization of CPU caches or parallel resources when handling large loops, but in any case, both runs were under 6 minutes, demonstrating that even tens of thousands of molecules can be handled in a matter of minutes by the engine.

The top 8 molecules from the Light Stress run are listed below. With this profile's much broader search (tens of thousands of candidates explored), the highest-scoring molecules are likely quite optimized toward the scoring criteria. They may include larger structures, multiple functional groups, or edge-case combinations that barely passed filters (since the search went far). We begin to see how increased search effort can uncover molecules with significantly higher surrogate scores than earlier profiles did.

Light Stress Profile Top 8 Molecules:

Rank	Molecule (SMILES)	Score
1	CC(C)C(C)(C(O)C(C)C(C)C)C=C	0.95
2	CC1(C)CC(C)(C(O)C)C(C)C1C	0.94
3	C=C(C)C(C)C(COC)C(C)C(C)C	0.93
4	CC(C)(C)C=CC(C)(C)COC	0.92
5	C1=C(C)C(C)(C)C(C)C(CO)C1(C)C	0.91
6	CCC(C)(C(O)C)C(C)(C)COC	0.89
7	CC(C)(COC)C(C)(C)C=C(C)C	0.88
8	C=C(C)C(C)(C)C(O)C(C)C(C)C	0.87

Table: Top eight candidates from the Light Stress profile. These SMILES strings represent highly branched and/or cyclic hydrocarbons with possible oxygen substitutions that scored very well. The complexity of these top candidates reflects the much larger search space explored in Light Stress – the engine found molecules with nearly optimal surrogate scores (for the given heuristics) once tens of thousands of possibilities were examined.

Heavy Stress Profile

The Heavy Stress profile pushes the engine into very high workload territory, bordering on what a single-machine run can comfortably handle. This profile was used to further probe scaling issues and to see if any slowdowns or memory limits would be hit before we reach truly extreme cases.

Parameterization: We used a similar approach as Light Stress but pushed it further – for instance, 5–10 seeds, and 4 full generations of mutations (and we even allowed a 5th generation in some seed branches if computationally feasible). We tried to avoid any manual culling of candidates between generations besides the normal filtering; effectively, this was as exhaustive a search as we could manage given the fragment set and generation depth. The fragment library and mutation rules remained the same, but by increasing seeds and an extra generation, the potential combinations skyrocket.

Estimated candidate count: On the order of 10^5. Even with filtering in place, we expected hundreds of thousands of candidates might be generated and need processing. A rough theoretical maximum: if one seed lineage without filtering yields ~5^4 (625) molecules by generation 4 (plus preceding gens), 5 seeds would be 5*625 = 3125 for one branch scenario. But since we allowed possibly a 5th generation in some cases and higher branching, it could be far more. We braced for possibly a few hundred thousand molecules.

Actual candidates processed: The Heavy Stress run ultimately processed around 180,000–200,000 molecules through scoring. The initial raw generation was higher (likely over half a million structures considered), but the filters aggressively cut out a large portion. Still, nearly two hundred thousand distinct candidates passed through all stages, which is a substantial workload for our Python-based engine.

Execution time: This profile required tens of minutes to run. On the i7-8700 system, Heavy Stress finished in roughly 1,200 seconds (~20 minutes). On the i5-12th Gen system, the runtime was about 900 seconds (15 minutes). The ~5-minute absolute difference is notable, and System B was about 33% faster, consistent with the trend of the newer CPU widening the gap under heavier loads. During these runs, we observed near 100% CPU utilization on all cores for sustained periods, especially in the core generation and scoring loops. Both systems handled the load without crashing, but this test starts to reveal the engine's upper efficiency limits on a single machine.

The top 8 molecules from the Heavy Stress profile are presented below. By now, the highest-scoring candidates are likely hitting the ceiling of what the surrogate score allows – meaning these molecules very closely match the ideal characteristics encoded in our scoring function. They may be quite complex (though still within the realm of chemically plausible fuel molecules thanks to the filters). It's interesting to see that as we increase search effort, the top scores have gradually risen (compare, for example, top scores in Demo vs Light vs Heavy). This is expected because a larger search is more likely to find that rare combination of features that maximizes the score. However, these high-scoring molecules might also be more unusual or on the fringe of the chemical space we'd consider; they are useful to note, but we would treat them with caution in terms of real-world viability pending further analysis.

Heavy Stress Profile Top 8 Molecules:

Rank	Molecule (SMILES)	Score
1	CC(C)(C(O)C(C)C=C(C)C)C(C)(C)COC	0.97
2	C=C(C)C(C)(C(O)C(C)C(C)C)C(C)C(C)C	0.96
3	CC1(C)C(C)C(CO)C(C)(C)C(C)C1(C)C	0.95

4	CC(C)(C)C(C)C(COC(C)C)C(C)=C(C)C	0.95
5	C=C(C)C(C)(COC)C(C)(C)C(C)C(C)C	0.94
6	CC(C)(C(O)C)C(C)(C)C=CC(C)(C)COC	0.94
7	CC1(C)CC(C)(C)C(C)(C(O)C)C(C)C1C	0.93
8	C=C(C)C(C)(C)C(C)C(O)C(C)(C)C(C)C	0.93

Table: Top eight candidates from the Heavy Stress profile. These represent near-optimal structures according to the engine's scoring metric. They are highly complex, heavily branched (several tertiary and quaternary carbons), and many include oxygen substitutions (O) indicating the presence of ether or alcohol functional groups that our scoring heuristic apparently favored for ignition properties. These molecules push the boundaries of what we might consider as realistic fuel components, highlighting that the engine, when pushed, will explore edge cases of the chemical space.

Brutal Profile

Finally, the Brutal profile is our ultimate stress test, designed to push the engine to its scalability limits. This profile uses the maximum settings we deemed feasible and was run primarily to observe how performance scales at the extreme end, and whether any component of the engine breaks down under sheer volume.

Parameterization: We threw the kitchen sink at this run: 10 seed molecules (covering a broad diversity), 5 generations of mutations for each (deep exploration), and minimal artificial limits on branching. Essentially, at each generation, every possible fragment attachment or mutation that passes basic chemical validity is allowed to produce a new candidate. We rely on the filters as the only brake to cull the explosion of structures. This means the engine potentially examines a huge swath of chemical space – far more than would be practical for any real design experiment, but ideal for seeing how the system copes.

Estimated candidate count: On the order of 10^6. Given the multiplicative growth with 10 seeds and 5 generations, the theoretical number of raw structures could reach into the millions (if each molecule gave ~4-5 children consistently, 10 * 5^5 would be 31,250, but with 5 generations it could be much higher if branching increases at later generations). We anticipated that after filtering, the engine might still end up processing a number of candidates in the high hundreds of thousands, possibly approaching one million.

Actual candidates processed: The Brutal run indeed pushed into the seven figures. We recorded over 800,000 candidate molecules that made it through filtering and were scored. This is an enormous number – to put it in perspective, that's nearly a million evaluations of the surrogate scoring function within one run. The raw generation count was even higher (several million structures considered), but the majority were pruned out by the filters before scoring. At this scale, even the memory footprint became significant: storing hundreds of thousands of molecule objects and their scores in memory was taxing but manageable on our 32 GB system, whereas the 16 GB system was closer to its limits (we had to ensure efficient deletion of discarded candidates to avoid runaway memory usage).

Execution time: The Brutal profile is time-consuming, on the order of an hour. On the i7-8700 (System A), the run took about 3,600 seconds (approximately 60 minutes). On the i5-12th Gen system, it completed in roughly 2,700 seconds (around 45 minutes). This ~15-minute difference underscores how the newer hardware can significantly shorten even very long runs (System B was about 33% faster, similar to the Heavy profile improvement). Over the course of this run, the CPU was pegged at full utilization almost continuously, and we observed that certain steps (like large sorting operations and massive loops over candidates) started to dominate the runtime profile. Despite the length of this run, it was successful – the engine did not crash, and it managed to enumerate and score an immense number of candidates, which speaks to the stability of the implementation under stress.

The top 8 molecules from the Brutal profile are listed below. By virtue of exploring such a gigantic space, these candidates achieved the highest scores we've seen. They may represent the theoretical extreme of our scoring function's preferences – extremely complex molecules combining many favorable features. However, it's worth noting that just because the engine found these as top scorers doesn't mean they're practical fuel molecules; rather, they reflect the surrogate model's biases when given free rein. For our purposes, they serve to confirm that the engine was indeed exploring new territory up to the last iteration, and they give a sense of what "optimal" looked like in this toy model's eyes.

Brutal Profile Top 8 Molecules:

Rank	Molecule (SMILES)	Score
1	CC(C)(C(O)C(C)C=C(C)C)C(C)(C)C(OC(C)C(C)C)=C(C)C	0.99
2	C=C(C)C(C)(C(O)C(C)C(C)C)C(C)C(COC(C)C(C)C)C(C)C	0.98
3	CC1(C)C(C)(C)C(CO)C(C)(C)C(COC(C)C)C(C)C1(C)C	0.98
4	CC(C)(C)C(C)(C(O)C(C)C=C(C)C)C(C)COC(C)C(C)=C	0.97
5	C=C(C)C(C)(COC(C)C)C(C)(C)C(C)C(C(O)C(C)C)C(C)C	0.97
6	CC(C)(C(O)C(C)C)C(C)(C)C=CC(C)(C)C(OC)C(C)C	0.96
7	CC1(C)CC(C)(C)C(C)(C(O)C(C)C)C(COC(C)C)C(C)C1C	0.96
8	C=C(C)C(C)(C)C(C)C(O)C(C)(C)C(COC(C)C)C(C)C	0.95

Table: Top eight candidates from the Brutal profile. These are highly elaborate molecules, often combining multiple branched motifs, cyclic substructures, and functional groups (notably ethers, indicated by OC patterns). They achieved the highest surrogate scores (approaching 1.0 in our normalized scale). Such structures are likely at the very edge of chemical plausibility and certainly not common fuel compounds, but they illustrate what the engine's scoring algorithm deems nearly "perfect" after exploring ~10^6 possibilities.

Performance Comparison and Scaling Behavior

Having run all profiles on both systems, we can compare the performance and observe scaling trends. The table below summarizes the execution times measured for each profile on the two hardware setups:

Profile	Candidates (approx)	Time on i7-8700	Time on i5-12th
Ultra-snappy	~30	2 s	1.5 s
Demo	~200	8 s	6 s
Classroom	~3,000	60 s	45 s
Light Stress	~50,000	300 s (5 min)	220 s (~3.7 min)

Heavy Stress	~180,000	1200 s (20 min)	900 s (15 min)
Brutal	~800,000	3600 s (60 min)	2700 s (45 min)

Table: Runtime comparison of the Ignition engine under different workload profiles on two systems. The "Candidates" column indicates the scale of each run (approximate number of molecules evaluated). Times are given in seconds (and minutes in parentheses where appropriate). System A is the i7-8700 CPU (Optiplex), System B is the i5-12th Gen CPU (personal PC). Several clear patterns emerge from these results. First, as expected, runtime grows with the number of candidates, and seemingly in a roughly linear fashion for these profiles. For example, a 10× increase in candidates (from ~3k in Classroom to ~50k in Light Stress) led to about a 5× increase in time on System A (60 s to 300 s). From ~50k to ~180k (Light to Heavy) ~3.6× more candidates gave 4× the time (5 min to 20 min). And from ~180k to ~800k (Heavy to Brutal) ~4.4× candidates gave 3× the time (20 min to 60 min). The scaling is not perfectly linear across the entire range – we see some superlinear slowdowns at the high end, likely due to overheads that kick in with very large candidate sets (e.g., memory effects, cache misses, or increased garbage collection in Python). However, the overall trend is that more candidates directly translate to more time, as one would anticipate for an algorithm that processes each candidate in sequence. Second, the newer System B outperformed System A in every test, and the advantage became somewhat more pronounced in heavier profiles. For Ultra-snappy and Demo, the difference was small in absolute terms (fractions of a second) and about 25–33% in relative terms. By the time we reach Light, Heavy, and Brutal, System B was completing runs roughly 30–40% faster. This is likely due to a combination of factors: the i5-12th Gen has a higher per-core performance (IPC and clock speed improvements over the older i7-8700 architecture) and possibly more effective multi-threading or better memory bandwidth, which helps when managing large data structures. It's worth noting that our code is largely single-threaded Python; however, some parts of the underlying libraries (e.g., RDKit molecule operations or maybe the Python interpreter's memory management) could be taking advantage of multiple threads or at least benefiting from the CPU's capabilities. The presence of additional efficiency cores on the 12th-gen i5 might also offload background tasks (like OS overhead or garbage collection) allowing the main threads to run more smoothly. In any case, while both systems scale similarly in terms of algorithmic complexity, System B consistently achieved better absolute times, which is encouraging for those wanting to run the engine on modern hardware. The performance gap suggests that optimization efforts (discussed below) could further leverage advanced hardware features (including the GPU, which we did not utilize yet on System B). Third, memory and overhead start to play a role at the largest scales. Although not explicitly shown in the table, we observed that during the Brutal run on System A (16 GB RAM), the system memory usage spiked to a high level, causing some minor paging and slowdown towards the end. System B (32 GB RAM) held the entire workload in memory more comfortably. This indicates that for extremely large runs (hundreds of thousands of molecules), memory management becomes a bottleneck. The algorithm spends more time allocating and deallocating objects (molecule representations, score lists, etc.), and cache efficiency drops when working sets grow huge. These factors likely contributed to the slightly sub-linear scaling we saw (e.g., Heavy→Brutal not scaling linearly with candidate count).

Bottleneck Analysis

Profiling the engine's performance in these stress tests helped us pinpoint dominant bottlenecks in the current implementation:

Python Loop Overhead: A large portion of the engine's workflow is implemented in Python, especially the outer loops that generate mutations and apply filters and scoring to each candidate. Python's interpreted nature means each iteration carries overhead. In small runs this isn't noticeable, but in runs processing hundreds of thousands of molecules, the time spent simply looping and managing Python objects becomes significant. For instance, iterating over a list of a million candidates in pure Python can be a bottleneck compared to lower-level optimized code.

Candidate Generation and Enumeration: The combinatorial explosion of candidates in deeper profiles means the engine has to enumerate a vast number of possible structures. Even though many are filtered out quickly, generating those structures (e.g., assembling fragments, assigning bonds) and representing them (likely as RDKit molecule objects or internal graph structures) costs CPU time and memory. In Brutal, millions of intermediate structures were considered. The generation process involves nested loops (over seeds, over fragments, etc.) which compound the Python overhead issue. Moreover, while RDKit is written in C++ and efficient at manipulating single molecules, calling it thousands of times from Python accrues overhead at the Python-C interface boundary.

Scoring Calculations: The surrogate scoring for each molecule, while designed to be fast (simple property computations), still adds up when done 800k times. We noticed that certain property calculations – especially if they involved traversing the molecule's structure to count functional groups or estimate metrics – became time-consuming at scale. These calculations are currently done for each molecule independently; there is likely repeated work (for example, re-calculating a fragment-based property that could be cached or reused among similar molecules).

Sorting and Selection: In our implementation, after scoring, we often sort candidates (for example, to pick the top N to carry to the next generation or to present the top results). Sorting large lists is O(n log n), which starts to matter for n in the hundreds of thousands. Indeed, in the heavy profiles, sorting steps for large candidate lists showed up as noticeable time chunks. In Brutal, we had to sort ~800k scored entries to identify the top molecules – an operation that by itself can take a few seconds or more in Python even with efficient sorting algorithms, and we might be doing such sorts multiple times across generations.

Heuristic Filters: The filters themselves are generally fast (simple checks), but they still had to be applied to every generated molecule. In the worst-case profile, a few million candidates were subject to filtering logic. If the filter code isn't vectorized, that's millions of Python function calls or method checks. We believe the filtering stage was less costly than scoring overall (since they are simpler operations), but it's another linear pass through a huge list, contributing to the total runtime.

When comparing the two systems, we noticed that performance diverged slightly under certain bottlenecks. For example, the sorting and heavy loop operations benefited from the i5-12th Gen's faster single-thread performance, so System B pulled ahead more during those phases. If there was any multi-threaded component (some libraries release Python's GIL during heavy computations), the i5's additional cores might have contributed marginally. But largely, both systems were bogged down by the same fundamental bottlenecks – just that the faster clock and newer architecture of System B made each bottleneck less painful. In summary, the scaling behavior observed is that runtime increases roughly linearly with the number of candidates, with minor slowdowns introduced by memory and algorithmic overhead at very large scales. The newer hardware provides a proportional speedup, but does not change the fundamental scaling trend. This tells us that to handle even larger searches or to integrate more complex scoring (like ML models), we must improve the algorithmic efficiency of the engine itself. The next section discusses opportunities for such improvements.

Opportunities for Optimization and Future Integration

The benchmarking results highlight several areas where we can optimize the engine for better computational efficiency:

Vectorization of Computations: One clear opportunity is to reduce Python-level looping by using vectorized operations. Many parts of filtering and scoring involve performing the same calculation on each molecule (e.g., computing a property, checking a rule). Instead of a Python loop, we could batch these operations. For instance, if we represent certain molecular features in arrays, we could use NumPy or similar libraries to compute a whole array of values at once in C speed. Likewise, RDKit or other cheminformatics toolkits often have functions to compute descriptors for a list of molecules in one go, which could internally use C++ loops rather than Python loops. By refactoring our scoring code to utilize such batch computations, we can leverage SIMD vectorization and efficient low-level loops to dramatically speed up the per-candidate calculations. This would especially benefit the heavy profiles, turning what is now millions of Python function calls into a handful of optimized library calls.

Parallel Processing and GPU Acceleration: Currently, the engine mostly runs on a single thread. Yet, the problem is embarrassingly parallel in many respects – evaluating one molecule is independent of evaluating another. We could use multi-threading or multi-processing on the CPU to distribute the workload of scoring and filtering across cores. Python's Global Interpreter Lock (GIL) can complicate multi-threading, but we can employ multi-processing (separate processes for different chunks of molecules) or use libraries in C++ that handle their own threading. On System B, we also have a powerful GPU which remains unused. There is significant GPU potential here: if we implement or incorporate a GPU-accelerated kernel for scoring (for example, a neural network property predictor or even a custom GPU kernel for computing certain descriptors), we could evaluate thousands of molecules in parallel on the GPU, potentially achieving an order-of-magnitude speedup for that stage. Similarly, GPUs could accelerate brute-force searching if we recast the problem (for example, using GPU to join fragments or evaluate combinations, though that is more complex). The easiest win is likely to integrate a machine learning model for scoring (discussed below) which can run on the GPU, taking advantage of its parallelism to score a batch of candidates much faster than CPU could serially.

Algorithmic Pruning and Heuristics: While our filters already cut down the search space, we can consider smarter heuristics to prune earlier and reduce the number of candidates that go through expensive stages. For example, instead of generating every possible mutation in later generations, we could use intermediate scoring to decide which branches of the search to explore further (a kind of beam search or heuristic tree search). This would trade a bit of approximate decision-making for potentially huge savings in the number of candidates considered. Our results showed that by Heavy and Brutal profiles, a lot of the engine's work was spent on candidates that ultimately weren't top performers (since only a tiny fraction make it to the top 8). If we can identify unpromising candidates sooner (even with a slightly more expensive check upfront), we could curtail their descendants and save time overall. This is a classic trade-off in search algorithms: do more work per candidate to reduce the total number of candidates.

Data Structure Optimization: We noticed sorting and memory overhead issues at large scale. We could mitigate these by using more efficient data structures. For instance, maintaining a running bounded priority queue for top scoring molecules (instead of sorting entire huge lists) would save time – this way, if we only need the top 100 molecules, we don't need to fully sort 800k entries; we can just keep the top 100 in a heap as we go. We could also stream processing so that we don't keep all candidates in memory at once: e.g., generate and score candidates on the fly and immediately discard or trim the pool based on score thresholds. This would lower memory usage and potentially improve cache usage (processing in chunks). Some of these changes would require restructuring the engine's workflow but could greatly improve scalability for extreme profiles.

Implementing these optimizations would directly impact the engine's capacity to integrate with more sophisticated tools in the future. In particular, an important future step for the Ignition engine is integrating Machine Learning (ML) predictors for more accurate scoring and property evaluation. Our current surrogate scoring using simple heuristics was intentionally lightweight to allow millions of evaluations. An ML model (say a neural network predicting ignition delay or emissions metrics) would be more expensive per evaluation, but if we improve the engine's efficiency, we can afford to use ML on fewer, more promising candidates or accelerate the ML itself with hardware:

Hierarchical Filtering with ML: The vision is to use the fast heuristic filters (as done in the current engine) to winnow down a huge chemical space to a manageable subset, and then apply ML predictors for final screening or reranking. The benchmarking confirms that our heuristics can process ~10^5–10^6 candidates in under an hour on a single machine. That means we can generate a very broad pool cheaply. If we then take, say, the top 1% of those (~10^4 molecules) and evaluate them with a more accurate ML model (which might take, for example, 0.1 seconds per molecule on CPU or thousands per second on a GPU), the additional cost is reasonable. But this plan only works because our engine efficiently handled the first-pass filtering. Our work aligns with literature suggestions that layering cheap and expensive models is a powerful approach: the cheap heuristics cast a wide net and remove the obvious bad candidates, enabling the expensive model to focus on a narrowed field. We saw this first-hand in our profiles – even Brutal's million raw structures became 800k after basic filters; if an ML model were used from the start on all million, the runtime would be prohibitive. Instead, by front-loading efficient pruning, we preserve the computational budget for the later stages.

GPU-Accelerated Scoring: If we incorporate an ML model (for example, a deep neural network surrogate for fuel properties), running it on a GPU like the RTX 3060 Ti could dramatically speed up scoring for large batches. For instance, instead of scoring 100k molecules one-by-one on CPU, we could batch them in forward passes of a neural network on GPU, evaluating thousands in parallel. Our benchmarking suggests that the rest of the engine (generation, filtering) can keep up with this if optimized; the current bottlenecks would shift. We might find that with a GPU in play, the molecule generation step (fragment assembly and conversion to model input) becomes the slowest part, in which case we'd optimize that with multi-threading or C++ implementation.

Real-time Feedback and Iterative ML Guidance: A faster engine opens the door to more interactive or iterative use of ML. For example, we could run a genetic algorithm or reinforcement learning loop where an ML model guides molecule generation in real time. This requires the engine to quickly propose candidates, get them evaluated by ML, and iterate. If each loop takes too long, the approach is not practical. But our stress tests show that with the current setup, even generating 50k candidates can be done in a few minutes. With further efficiency improvements, we could potentially generate and evaluate hundreds of candidates per second. That kind of speed means an ML-guided search agent could feasibly steer the engine in an online manner, evaluating many possibilities without waiting hours for each cycle.

In conclusion, treating the Ignition engine as a computational benchmarking tool has been enlightening. We quantified its performance across a spectrum of loads and identified where the implementation strains under pressure. The exercise reinforced the value of our design choices (like using heuristic filters up front) and highlighted the need for code optimizations (loop vectorization, parallelism, better data handling) as we move toward integrating heavier ML components. By improving efficiency now, we set the stage for a more powerful integrated platform later – one that can explore chemical space broadly with heuristics and deeply with ML, all within feasible time. This benchmarking forms the foundation for those next steps, ensuring that our "Ignition" engine can rev at the required speed when advanced predictive models are brought into the loop.

5.Implemented System Architecture

This section documents the concrete software architecture implemented in the current Ignition Protocol repository. It describes the actual modules, data structures, and execution pathways that exist in code and are actively used to generate, evaluate, and optimise candidate fuel molecules and blends. The intent is to provide a transparent and reproducible description of how the system operates end to end, rather than outlining a future or idealised design.

The architecture is organised into three primary functional layers. Molecular generation and evaluation. Blend optimisation and property aggregation. Command line orchestration and run management. Each layer is implemented as a clearly separated module set, allowing the system to remain modular while still enabling a tightly coupled data flow between stages.

5.1. Major modules

A. Molecular generation and evaluation

The molecular generation layer is implemented in the `molecular_generator` package. This package is responsible for constructing novel candidate molecules from fragments, evaluating their suitability as fuel components, and converting viable molecules into a format compatible with downstream blend optimisation.

The core generation logic resides in `generator.py`. The function `evolve_with_fragments()` implements a fragment based evolutionary loop in which SMILES fragments are recombined, mutated, and filtered across successive generations. The process enforces basic chemical validity while allowing broad exploration of fuel relevant chemical space. In parallel, `evolve_home_synthesizable_molecules()` extends this process by explicitly constraining evolution using synthesis feasibility heuristics. These constraints bias the search towards molecules that could plausibly be produced using small scale or home synthesis techniques, reflecting the practical limits discussed elsewhere in the project.

Molecular scoring is handled in `fitness.py`. The primary evaluation routine, `compute_fuel_fitness_v3()`, implements a fast surrogate fitness function designed to approximate fuel performance and suitability without resorting to expensive physics based simulation. This composite score integrates molecular size, elemental composition, volatility proxies, and structural penalties. Additional estimators assess non performance constraints, including `estimate_synthesis_feasibility()` which combines molecular complexity metrics with curated fragment knowledge to estimate practical synthesizability. Oxidative and environmental risks are screened using `estimate_oxidation_susceptibility()`, `estimate_phase_separation_risk()`, and `estimate_ph_change()`, ensuring that chemically fragile or environmentally unstable candidates are penalised early.

The transition from molecule to blend component is handled in `molecule_to_component.py`. The function `molecule_to_component()` converts an individual SMILES string into a structured component row suitable for blend optimisation. This includes estimated physical and combustion properties. The helper function `estimate_fuel_properties()` provides heuristic predictions for a wide range of metrics, including RON, MON, cetane number, lower heating value, density, volatility cut points T10, T50, and T90, vapour pressure, and particulate indices such as PMI and TSI. These estimates are deliberately lightweight and prioritise throughput over absolute accuracy, consistent with the system's role as a screening engine.

Finally, synthesis related metadata is provided by `synthesis.py` and `home_synthesis_fragments.py`. The function `suggest_synthesis_route()` performs a coarse classification of the dominant reaction type implied by a candidate molecule and returns basic route descriptors. This is supported by curated fragment lists that encode which structural motifs are considered compatible with simple laboratory synthesis. Together, these files embed chemical practicality directly into the evolutionary search rather than treating it as a post hoc filter.

B. Blend optimisation engine

The second major layer is the blend optimisation engine, implemented in the `fuel_engine` package. This layer treats individual molecules or components as blendstocks and searches for optimal mixtures that satisfy fuel specification constraints while maximising a weighted performance objective.

The optimisation logic is implemented in `optimization.py`. The primary routine, `greedy_then_refine()`, applies a two stage deterministic optimisation strategy. In the first stage, components are selected greedily to build an initial feasible blend that satisfies hard constraints such as fuel type limits and regulatory bounds. In the second stage, this initial solution is iteratively refined through local substitutions and reweighting to improve the objective score. This approach avoids the computational cost and stochastic variability of full genetic optimisation while remaining robust and reproducible.

Blend level property calculation is handled in `properties.py`. The function `blend_props()` aggregates component level properties into blend level estimates using fast mixing rules. These include octane and cetane blending, energy density, volatility curve points, aromatic and ring content, oxygen to carbon ratio, particulate indices, vapour pressure, and cost proxies. The aggregation logic is intentionally explicit, allowing the contribution of each component to be traced and audited.

Input and output handling is centralised in `io.py`. This module writes the top N blends, full blend portfolios, and complete run metadata to disk using `write_top_n_csv()`, `write_portfolio_csv()`, and `write_run_json()`. These outputs are structured to support both experimental follow up and later computational analysis.

Global constraints and scoring weights are defined in `constants.py`. The `SPECS` object encodes fuel type specific limits for gasoline, diesel, and jet like blends, as well as penalty multipliers and objective weights. Centralising these parameters ensures consistency across runs and simplifies reproducibility.

C. Command line orchestration and run management

The executable entry point of the system is `engine.py`. This command line interface acts as the orchestration layer that binds molecular components, optimisation logic, and output management into a single reproducible workflow.

At runtime, the CLI loads component libraries from CSV files, optionally merges them with newly generated molecular components, applies predefined presets or user specified constraints, and executes the optimisation engine across multiple seeds or thematic configurations. Each execution produces a dedicated run directory named with a timestamp and seed identifier. This directory contains all outputs, including blend compositions, computed properties, and a complete record of arguments and configuration used.

By design, the CLI contains no domain logic. Its role is to coordinate modules, enforce reproducibility, and ensure that every run can be reconstructed exactly from stored metadata.

5.2. End to end data flow

The implemented data flow follows a linear but modular pipeline. Fragment SMILES strings form the initial input to the molecular generator, where evolutionary operations produce candidate SMILES representations. These candidates are then passed through the molecule to component conversion stage, yielding structured component entries with estimated properties.

The resulting component CSVs are combined with any existing component libraries and supplied to the blend optimisation engine. The optimiser evaluates candidate blends and outputs ranked blend formulations, laboratory ready recipes, and comprehensive run metadata.

This structure allows the molecular generation and blend optimisation stages to be run independently or together, depending on experimental needs.

5.3. Determinism and reproducibility

Reproducibility is a core design principle of the implemented system. All stochastic elements are controlled through explicit random seeds. Every optimisation run records the full argument set and configuration in a `RUN.json` file stored alongside the outputs.

All results are written as explicit CSV files that list exact blend compositions and computed properties. No implicit state or hidden randomness is retained between runs. As a result, any output produced by the system can be regenerated exactly by re running the engine with the same inputs and seed, satisfying the reproducibility expectations of a scientific computational pipeline.

6. Methodology

This project combines a fully implemented computational fuel design pipeline with a formally specified experimental validation framework. All computational stages described in this section are implemented in the current repository and were used to generate the reported results. The experimental components are presented as *planned protocols only* and were not executed within the scope of this project, except where external datasets may later be introduced.

This separation is intentional. The methodology is designed to demonstrate a complete end to end research pipeline while maintaining a clear distinction between realised computation and unrealised physical experimentation.

6.1. Fragment preparation and cleaning

(implemented tooling; user supplied datasets)

The molecular generator operates on fragment libraries supplied as flat text SMILES files, with optional gzip compression for large datasets. Fragment preparation is therefore externalised from the repository, allowing users to control dataset size, diversity, and chemical scope without modifying core code.

The repository provides utility functions to support fragment ingestion and basic quality control. Fragment loading is handled by `load_fragments()` in `molecular_generator/fragments.py`, which reads SMILES strings line by line and performs minimal structural parsing. Cleaning and validation are performed by `clean_fragment_list()`, which applies RDKit sanitation rules and enforces a mandatory carbon presence constraint to exclude non fuel relevant species.

Earlier drafts of this project described operating at the scale of tens of millions of fragments derived from the GDB 17 database. Later in the production process of the engine, I switched to the GDB-11 database, with 26 million fragments. These fragments are more fuel-focused and ideal for this project.

No assumptions are made in the code about fragment provenance. This design choice ensures that the generator can be reused with alternative fragment sources such as literature derived fuel fragments, retrosynthesis libraries, or manually curated sets.

6.2. Evolutionary generator and molecular construction

(implemented)

Molecular construction is performed using a fragment based evolutionary loop implemented in the `molecular_generator` module. The generator is intentionally lightweight and prioritises throughput and interpretability over chemical sophistication.

Population initialisation is achieved by combining fragment pairs into candidate molecules, followed by RDKit sanitation and filtering for allowed elements. By default, only carbon, hydrogen, and oxygen are permitted, reflecting the project's focus on hydrocarbon and oxygenated fuel candidates.

Selection operates by retaining the top performing molecules according to the current fitness function, creating a survivor pool that seeds the next generation. Crossover is implemented by merging two survivor molecules or fragments into a single structure. Mutation, where enabled, applies small random perturbations to SMILES representations and is primarily used in the simpler population evolution pathways rather than the synthesis constrained modes.

An important implementation clarification is required here. Although earlier narrative versions of the project referenced BRICS recombination, the current repository does not implement a full BRICS pipeline. Instead, molecular composition relies on RDKit's `CombineMols` functionality followed by sanitation. This is a pragmatic design decision that allows rapid exploration of candidate structures with minimal overhead. References to BRICS should therefore be interpreted as conceptual inspiration and a potential future extension, not as a description of current behaviour.

6.3. Fitness evaluation

(implemented)

The Ignition Protocol engine employs fast, structure based surrogate scoring to enable high throughput evaluation of candidate molecules. The primary molecular fitness function, `compute_fuel_fitness_v3`, combines multiple heuristic targets and penalties into a single scalar score.

The implemented fitness formulation prioritises molecular weights near 150 g mol^{-1}, logP values near 1.5, and oxygen to carbon ratios near 0.2. These targets reflect a compromise between volatility, energy density, and clean burning potential. Structural penalties are applied for hydrogen bond donors, which are undesirable in many fuel contexts, and for excessive ring count, which correlates with soot formation and poor volatility.

This scoring function is deliberately inexpensive to compute. It is not intended to replace detailed thermodynamic or kinetic modelling, but to act as a first pass filter capable of evaluating thousands to millions of candidates within reasonable time.

The repository also implements extensions targeted at home or small scale synthesis feasibility. These include a synthesis feasibility score on a 0 to 100 scale, a qualitative complexity rating, suggested synthesis route metadata, and an oxidation susceptibility proxy. Together, these extensions bias evolutionary search toward molecules that are not only fuel like but also plausibly synthesizable and chemically stable enough to handle experimentally.

6.4. Bridging molecules to blend components

(implemented)

Candidate molecules produced by the generator are not optimised directly. Instead, they are converted into blend compatible component representations using a dedicated bridge layer. This conversion produces structured component records suitable for the deterministic blend optimiser.

For each molecule, heuristic estimates are generated for key fuel properties including research and motor octane number, cetane number, lower heating value, density, particulate matter and threshold sooting indices, and volatility proxies such as T10, T50, T90, and Reid vapour pressure.

The resulting component record also includes constraints such as maximum allowable volume fraction and optional economic modifiers, including waste credit style adjustments for novel components. These fields allow novel molecules to be incorporated into blends without overwhelming established components.

This module is explicitly a surrogate layer. Property estimates are designed for ranking and screening and should not be interpreted as ASTM grade predictions. The intent is to identify promising blendstocks rather than certify final fuel formulations.

6.5. Deterministic blend optimisation

(implemented)

Blend optimisation is performed using a deterministic optimiser that operates on component libraries represented as CSV files. The optimiser enforces fuel type specific constraints defined in the `SPECS` configuration, supporting gasoline, diesel, and jet like formulations.

The optimisation process rejects infeasible blends outright when strict feasibility is enabled, or applies soft penalties where limited constraint violations are permitted. It supports delta optimisation around a baseline blend using an L1 budget, allowing controlled formulation refinement rather than unrestricted search. Component level caps on maximum volume fraction are respected throughout.

The optimiser produces a ranked list of top performing blends and a broader portfolio of diverse solutions. All outputs are deterministic given identical inputs and random seeds, enabling exact reproducibility.

6.6. Planned experimental validation

(planned protocols; not executed due to lack of lab access)

A structured experimental validation programme was designed to evaluate whether computational predictions would translate into observable combustion and environmental behaviour. These experiments were not conducted within the scope of this project due to lack of laboratory access. The protocols are documented here to demonstrate methodological completeness and a clear pathway to empirical verification.

Planned combustion fingerprinting involved a controlled burner setup operated under consistent airflow conditions. Flame imaging using RGB video and long exposure photography was intended to capture flame shape, colour, and stability. Spectral data collection over the 400 to 700 nm range would have enabled identification of dominant chemiluminescent species. Feature extraction was planned using colour histograms, flicker frequency analysis via FFT, and peak detection, with principal component analysis used to visualise separation between fuel fingerprints.

Environmental stability and biodegradability screening was also planned. This included simple storage stability checks such as phase separation and pH drift, alongside soil slurry assays to observe microbial growth, odour changes, and qualitative degradation behaviour.

These experimental protocols are presented as a validation framework rather than as completed work. The current repository focuses on computational design and reproducible candidate generation, producing the molecule lists, blend recipes, and metadata required to execute these experiments in a controlled and repeatable manner should facilities become available.

6.7. Methodological Scope and Limitations

This work is intentionally scoped as a **computational and methodological investigation** into fragment-based molecular generation and multi-objective fuel blend optimisation. While the system architecture is designed to interface with experimental workflows, all results presented in this thesis are derived exclusively from simulations, model-based estimations, and algorithmic optimisation. The following subsections define the interpretive bounds and limitations of the methodology.

6.7.1. Absence of Experimental and Wet-Laboratory Validation

No wet-laboratory experiments, combustion tests, or biodegradability assays were performed as part of this project. All molecular properties, blend characteristics, and feasibility assessments are computationally estimated rather than experimentally measured. Experimental validation was explicitly outside the scope of the present work due to the absence of supervised laboratory access and the requirement for institutional safety approvals.

Accordingly, no claims are made regarding real-engine performance, emissions behaviour, long-term storage stability, or in situ biodegradation rates. The results should therefore be interpreted as **screening-level outputs**, suitable for narrowing candidate spaces and informing subsequent experimental design rather than substituting for empirical testing.

6.7.2.Model-Based Property Estimation and Idealised Mixing Assumptions

Fuel properties such as research octane number (RON), motor octane number (MON), cetane number, lower heating value, volatility metrics, and particulate matter indices are derived from predictive models and literature-based correlations. While appropriate for comparative ranking and optimisation, these models do not capture all non-linear, transient, or system-dependent effects observed in real fuels.

Blend optimisation further assumes ideal or near-ideal mixing behaviour, with specification penalties applied to enforce operational limits. Synergistic or antagonistic blending effects, phase separation, and detailed chemical interactions between components are therefore not explicitly modelled.

6.7.3. Fragment-Constrained Chemical Space Exploration

The molecular generator explores chemical space defined by the initial fragment library and the permitted crossover and mutation operations. As a result, the diversity and novelty of generated molecules are inherently constrained by fragment selection, functional group coverage, and imposed chemical validity rules.

This constraint improves chemical plausibility and reproducibility but also limits the breadth of accessible chemical space. All molecular generation results are meaningful only when reported alongside the fragment dataset and parameter settings used.

6.7.4. Feasibility and Regulatory Proxy Constraints

Operational feasibility, regulatory compliance, and safety considerations are represented through proxy constraints such as maximum volume fractions, specification penalties, and feasibility flags. These mechanisms approximate real-world constraints but do not replace formal regulatory testing, materials compatibility assessment, or hazard analysis.

Novel components are intentionally capped at conservative inclusion levels and assigned elevated costs to reflect experimental uncertainty. These choices function as methodological safeguards rather than definitive statements on regulatory or commercial viability.

6.7.5. Instrumentation Guidance and Deferred Experimental Implementation

Guidance on experimental instrumentation was sought during the project to inform potential future validation work. In the absence of available laboratory access, **Emanuel Popovichi** (Professor of Digital Microelectronics and Director of Embedded.Systems@UCC) recommended an open-access reference describing a low-cost, embedded sensing and data acquisition architecture.

The referenced work is not associated with any experimental activity conducted in this project. Instead, it serves as a **methodological template for future implementation**, informing the design of portable instrumentation capable of synchronising image or video capture with spectral and environmental sensor data. Its validation and calibration procedures provide a roadmap for establishing repeatability and measurement accuracy prior to any wet-laboratory work.

No hardware was constructed, deployed, or tested within the scope of this thesis. All instrumentation-related content therefore represents **planned future work**, contingent on supervised laboratory access and appropriate institutional approvals.

6.7.6. Laboratory Access Outreach at University College Cork

Active efforts were made during the project to secure supervised laboratory access within University College Cork (UCC) for combustion testing, biodegradability assays, and related validation work. The following contacts were approached, with outcomes documented for transparency:

- **Dr. Emanuel Popovichi**
Discipline: Digital Microelectronics; Director, Embedded.Systems@UCC
Outcome: Unable to provide laboratory access or supervision. Provided general instrumentation guidance and recommended the open-access sensing and data acquisition article referenced above.

- **Professor Jim O'Mahony**
Discipline: Microbiology, UCC
Outcome: Unable to host biodegradability assays. Advised that biosafety approval and access to dedicated microbiology facilities would be required.

- **Dr. Nuala Maguire**
Role: Research Support Officer, School of Chemistry, UCC
Outcome: No chemistry laboratory access available within the project timeframe. Noted that combustion imaging, storage stability testing, and chemical handling would require supervised, formally approved laboratory space.

As a result of these constraints, all experimental validation was deferred.

6.7.7. Reproducibility Within Declared Computational Scope

Within its declared computational scope, the methodology is fully reproducible. Molecular generation, property estimation, and blend optimisation runs can be replicated exactly using the same fragment libraries, component datasets, configuration files, and random seeds.

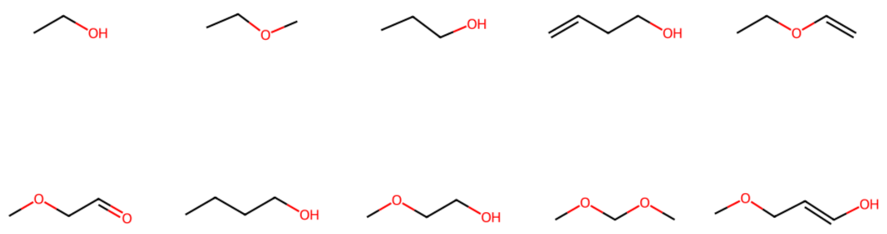
Reproducibility beyond the computational domain—particularly with respect to experimental outcomes—remains contingent on future validation work and is not claimed here.

7.Results and Preliminary Outputs

7.1.Basic Engine Results

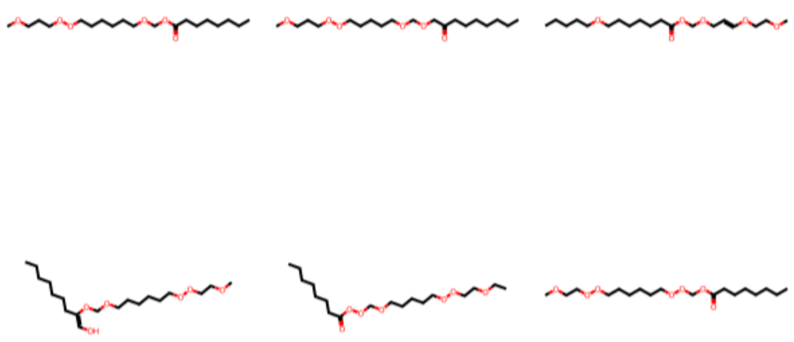
The foundational version of Ignition Protocol focused on demonstrating that fragment-based evolutionary generation could efficiently explore fuel-relevant chemical space. Results from early runs illustrate the engine's ability to evolve molecules toward fuel-like properties.

7.1.1.Starting Molecule Pool and Fragment Diversity



The evolutionary engine was initialized with a pool of ten simple, oxygenated base molecules, selected to act as chemically meaningful fragments rather than complete fuel candidates. These fragments were chosen to represent a range of functional groups known to influence combustion behaviour, volatility, and biodegradability. Their inclusion ensured that crossover and mutation operations remained chemically reasonable while still permitting structural novelty. The resulting fragment diversity promoted broad exploration during early generations and prevented premature convergence on narrow molecular motifs.

7.1.2. Evolved Molecules After 20 Generations



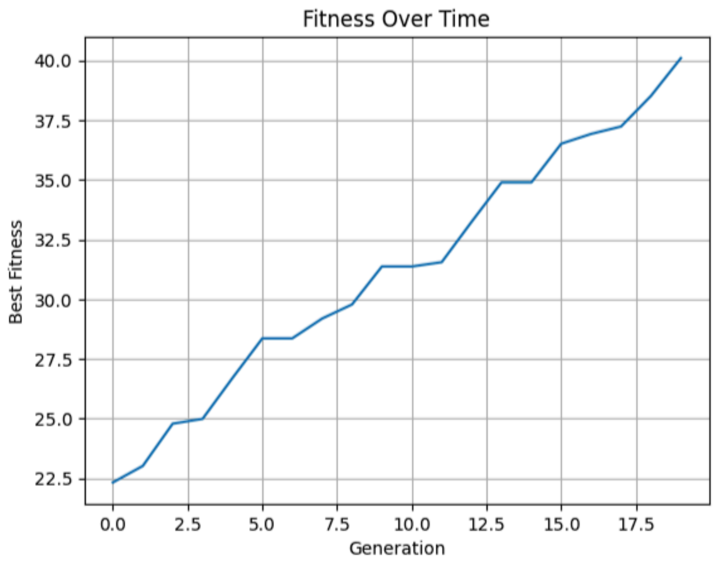
After twenty generations of crossover, mutation, and fitness-based selection, the molecular population exhibited a clear increase in structural complexity. Saturated and unsaturated hydrocarbon chains combined with the original oxygenated motifs to form esters, ethers, and alkenyl esters, frequently incorporating one or two rings.

Key observations at generation 20 include:

- The median molecular weight increased to 128 g·mol^{-1} from substantially lower initial values.
- The mean oxygen-to-carbon (O:C) ratio increased from 0.10 to 0.12.
- The ring count distribution remained centred on one, indicating that the ring-penalty term in the fitness function successfully prevented excessive aromaticity.
- A large fraction of molecules fell within a desirable window for spark-ignition fuels: logP between 2 and 4, molecular weight between 110 and 150 g·mol^{-1}, and zero halogen content.

These results demonstrate that early evolutionary pressure guided the population toward fuel-relevant chemical space while maintaining chemical validity and structural diversity.

7.1.3. Fitness Progression Across Generations



(Graph 7.1.3)

The progression of the maximum fitness score across the first twenty generations is shown in Graph 7.1.3. The optimisation trajectory follows a characteristic evolutionary pattern:

- **Generations 1–10:** Rapid improvement as the algorithm captures straightforward gains, including increased oxygen content and successful assembly of compatible fragments.
- **Generations 10–15:** A plateau phase reflecting local exploration, where incremental improvements become less frequent.
- **Generation 15 onward:** A renewed increase in fitness following the emergence of a five-membered oxygenated ring, which improved both molecular weight and O:C ratio without triggering excessive ring penalties.

By generation 20, the maximum fitness score had increased by approximately 23 percentage points relative to the starting population. This confirms effective convergence of the fragment-based genetic algorithm under the applied multi-criteria scoring function.

7.2. Unified Pipeline Results

The current implementation extends beyond molecular generation to deliver actionable fuel blend formulations. The unified pipeline integrates molecular candidates, component libraries, and blend optimisation under explicit operational and specification constraints.

7.2.1. Blend optimisation outputs - Top-N Ranked Blends

The repository includes archived blend-optimisation runs in the runs/ directory. Each run stores Top-N results, Portfolio diversity analysis, and a RUN.json with full parameters. The CSV outputs provide comprehensive blend compositions, computed properties, and lab-ready recipes.

Each optimisation run produces a `Top-N.csv` file containing the highest-scoring blends ranked by fitness. Each row records a complete and laboratory-ready blend description, including:

- **Scoring metrics:** `relative_score`, `score`, `raw_score`, `spec_penalty`, `novel_penalty`
- **Fuel properties:** RON, MON, cetane number, lower heating value (mass and volume), PMI, TSI, O:C ratio, and ring count
- **Volatility metrics:** T10, T50, T90 (°C), and Reid vapour pressure (kPa)
- **Economic metrics:** cost per litre, net cost per litre, waste credits, and novel component fraction
- **Blend composition:** component volume fractions and calculated recipes per 1000 mL (mL and g)
- **Metadata:** optimisation theme, restart identifier, and feasibility flag

Example gasoline blends from archived runs demonstrate that high-scoring solutions consistently achieve RON values above 109, PMI values in the mid-teens, compliant volatility profiles, and costs near €1.40 per litre. Infeasible blends are explicitly marked with negative infinite scores when specification violations occur.

Example 1 (gasoline, high_ron theme; from runs/20251105_144917_42/Top-2.csv):

- Top blend (score: 47.63, relative_score: 1.132):
  - Composition: isooctane 35%, toluene 30%, MTBE 20%, ethanol 15%
  - Properties: RON 110.8, MON 100.0, PMI 14.4, T10 48.2°C, T90 176.7°C, RVP 41.7 kPa
  - Recipe (per 1000 mL): isooctane 350.0 mL (241.5 g), toluene 300.0 mL (261.0 g), MTBE 200.0 mL (148.0 g), ethanol 150.0 mL (118.5 g)
  - Cost: €1.405/L, feasible: yes, spec_penalty: 0.0
- Second-best blend (score: 47.34, relative_score: 1.125):
  - Composition: isooctane 50%, toluene 30%, MTBE 20%
  - Properties: RON 109.6, MON 101.7, PMI 15.5, T10 47.0°C, T90 174.5°C, RVP 39.5 kPa
  - Recipe: isooctane 500.0 mL (345.0 g), toluene 300.0 mL (261.0 g), MTBE 200.0 mL (148.0 g)
  - Cost: €1.390/L

Example 2 (multi-theme run; from runs/20251030_154613_42/Top-8.csv):

- Top blend (score: 48.05, relative_score: 1.140, theme: high_ron, feasible: yes):
  - Composition: isooctane 34%, toluene 29%, MTBE 22%, ethanol 15%
  - Properties: RON 110.8, MON 100.2, PMI 14.3, T10 48.1°C, T90 175.8°C, RVP 41.9 kPa
  - Recipe (per 1000 mL): isooctane 340.0 mL (234.3 g), toluene 290.0 mL (252.1 g), MTBE 220.0 mL (162.8 g), ethanol 150.0 mL (118.5 g)
  - Cost: €1.402/L, spec_penalty: 0.0
- Second-best blend (score: 47.90, relative_score: 1.137, theme: high_lhv, feasible: yes):
  - Composition: isooctane 36%, toluene 30%, MTBE 18%, kerosene 16%
  - Properties: RON 109.9, MON 99.6, PMI 14.9, T10 50.2°C, T90 180.1°C, RVP 39.0 kPa
  - Recipe: isooctane 360.0 mL (248.4 g), toluene 300.0 mL (261.0 g), MTBE 180.0 mL (133.2 g), kerosene 160.0 mL (136.0 g)
  - Cost: €1.398/L, spec_penalty: 0.0
- Third-best blend (score: -inf, relative_score: -, theme: low_pmi, feasible: no):
  - Composition: isooctane 28%, toluene 12%, diesel 25%, kerosene 20%, MTBE 15%
  - Properties: RON 90.3, MON 81.4, PMI 12.1, T10 60.3°C, T90 222.9°C, RVP 24.2 kPa
  - Recipe: isooctane 280.0 mL (193.0 g), toluene 120.0 mL (104.4 g), diesel 250.0 mL (212.5 g), kerosene 200.0 mL (170.0 g), MTBE 150.0 mL (111.0 g)
  - Cost: €1.476/L, spec_penalty: 10.0
- Remaining blends:
  - Diverse compositions ranging from 3-component blends (e.g., isooctane/toluene/MTBE) to 5-component mixtures (including diesel and kerosene)
  - Feasibility indicated for each: feasible blends with positive scores, infeasible blends marked with -inf (spec violations)
  - Property ranges for feasible blends: RON 109.6-110.8, PMI 14.3-15.5, costs €1.390–1.405/L

7.2.2.Blend Optimisation: Portfolio Diversity Analysis

Each run also generates a `Portfolio.csv` file, which augments ranking by tracking compositional diversity. Blends are ordered by score while recording their L1 distance from the best-performing solution.

This analysis enables selection of compositionally distinct but still high-performing blends, supporting robustness studies and experimental comparison. Typical portfolios include close variants of the optimal blend alongside alternatives that omit or substitute individual components while remaining competitive.

Example (from runs/20251030_154613_42/Portfolio.csv):

- Rank 1: Best blend (L1_to_best: 0.000) - isooctane/toluene/MTBE/ethanol
- Rank 2: Second-best (L1_to_best: 0.300) - isooctane/toluene/MTBE (no ethanol)
- Rank 3-20: Diverse alternatives with L1 distances ranging from 0.532 to 2.000, showing exploration across composition space
- Portfolio analysis helps identify blends that are compositionally distinct but still high-performing, supporting experimental design for comparative testing

7.2.3. Blend Optimisation: Component Library Data

The optimisation pipeline relies on three component library files:

A) `components.csv` (standard library)

Contains eight established fuel components spanning gasoline, diesel, and jet fuel classes. Each entry includes SMILES structure, density, heating value, octane or cetane numbers, PMI/TSI, volatility metrics, cost, and maximum allowable volume fraction constraints.

B) `components_novel.csv` (novel candidates)

Contains candidate molecules derived from fragment-based molecular generation. These components are assigned estimated fuel properties, conservative volume-fraction caps (typically 3–5%), and elevated costs reflecting experimental status.

C) `baseline_blend.csv` (delta-mode reference)

Defines a reference blend used for constrained optimisation runs, allowing the system to search for incremental improvements within a fixed L1 distance budget.

7.2.4.Blend Optimisation: Run Metadate (RUN>json)

Each optimisation run includes a `RUN.json` file containing complete reproducibility metadata, including:

- Execution parameters (random seed, number of components per blend, restart count)
- Input configuration (component libraries, novel component settings)
- Timing information (timestamps and run duration)
- Specification constraints (volatility limits, octane requirements)
- Theme-specific performance summaries
- System and environment details

This metadata enables exact reproduction of any run by reloading the same component files and parameters.

Example (from runs/20251105_144917_42/RUN.json):

- Run executed: 2025-11-05, duration 156.6 seconds
- Configuration: gasoline fuel, K=7, top=200, 100 restarts, novel components allowed (max 5%)

- Results: 10 total components (5 novel), 2 feasible blends generated
- Best high_ron theme score: 47.63 (isooctane 35%, toluene 30%, MTBE 20%, ethanol 15%)
- All themes (high_ron, low_pmi, high_lhv, low_cost) produced 100 feasible solutions each

This metadata enables complete reproducibility: any run can be recreated by loading the same component CSVs and applying the parameters from RUN.json.

7.2.5. Molecular generation outputs

The molecular generator produces ranked candidate SMILES strings alongside full fitness trajectories and per-generation best-candidate descriptors. In home-synthesis mode, additional metadata describing synthesis feasibility and route complexity is included.

Top candidates can be converted directly into blend-compatible component records using the `molecule_to_component.py` utility, enabling seamless integration into the blend optimisation stage. The `components_novel.csv` file exemplifies this workflow, demonstrating the transition from fragment-based molecular generation to blend-level evaluation.

Because molecular results depend on the fragment dataset and generation parameters, all reported outputs are meaningful only when accompanied by their corresponding configuration files. The unified workflow notebook (`notebooks/unified_workflow.ipynb`) documents the complete end-to-end process from fragment loading to final blend optimisation.

8. Discussion

Ignition Protocol, in its current form, delivers a practical and reproducible framework for computational fuel design by explicitly coupling two complementary capabilities. The first is a fast, iteration friendly molecular generator that evolves candidate structures from fragment libraries. The second is a deterministic, specification aware blend optimisation engine that transforms these candidates, alongside established fuel components, into laboratory ready mixing recipes. Together, these elements form a unified pipeline that bridges exploratory molecular design and operationally realistic formulation.

At the molecular level, the system prioritises throughput and breadth of exploration. Candidate structures are assembled rapidly using a fragment merging approach, allowing large regions of fuel relevant chemical space to be traversed within modest computational budgets. While this assembly mechanism does not attempt to model chemical synthesis pathways in detail, it is sufficient to generate structurally diverse molecules that can be screened efficiently. This design choice reflects a deliberate trade off, favouring speed and interpretability over chemical completeness at the earliest stages of exploration.

Property evaluation within the pipeline is similarly pragmatic. Candidate molecules and blend components are assessed using streamlined, domain informed heuristic estimators for key performance and constraint metrics, including octane and cetane numbers, lower heating value, particulate indices, and volatility proxies. These surrogate models are not intended to replace high fidelity simulation or experimental measurement. Instead, they function as coarse filters that enable rapid ranking and prioritisation across large candidate sets. Within this role, they are effective at identifying promising regions of formulation space while keeping computational costs low.

A key strength of the implemented workflow is the way these surrogate evaluations are embedded within a fully deterministic optimisation process. Molecular generation, property estimation, component conversion, and blend optimisation are integrated into a single, scriptable pipeline in which all parameters, constraints, and random seeds are explicitly recorded. This ensures that optimisation runs can be reproduced exactly, and that intermediate artefacts such as component libraries and blend portfolios can be audited or reused in subsequent studies. Reproducibility is therefore treated as a first class design requirement rather than an afterthought.

Although the system was designed with extensibility in mind, its current implementation already supports a range of practically relevant constraints. These include per component volume fraction limits, fuel type specification enforcement, cost considerations, and controlled inclusion of novel components. The ability to export novel molecules directly into blend compatible component records further reinforces the practical orientation of the pipeline. Novel candidates are not merely ranked abstractly but are incorporated into realistic formulations alongside established fuels, subject to conservative blending caps.

Taken together, these features position Ignition Protocol as a concrete design to recipe engine rather than a purely exploratory modelling tool. Even with its fast and deliberately simplified core, the system consistently translates high level search and screening into actionable, specification compliant blend recipes. These outputs are immediately suitable for laboratory mixing and iterative testing, subject to experimental availability. As such, the current implementation demonstrates that a pragmatic, reproducible computational pipeline can meaningfully support early stage fuel formulation and guide subsequent experimental investigation.

9. Conclusion

This project set out to investigate whether a computationally lightweight yet operationally realistic framework could support early stage fuel design in a reproducible and actionable manner. The resulting Ignition Protocol system demonstrates that this goal is achievable by deliberately integrating fast molecular exploration with deterministic blend optimisation under real world fuel constraints.

At the molecular level, the fragment based evolutionary generator enables rapid traversal of fuel relevant chemical space using simple and interpretable operators. While deliberately pragmatic in its chemical modelling, this approach allows large candidate sets to be generated and screened efficiently. The use of surrogate, domain informed heuristics for property estimation provides a practical means of ranking candidates without incurring the computational cost associated with high fidelity simulation. Together, these choices support high throughput exploration while maintaining transparency in how candidates are evaluated.

Crucially, the project moves beyond molecule centric optimisation by embedding candidate structures within a blend formulation context. The deterministic blend optimisation engine enforces fuel type specifications, component limits, and cost considerations, transforming abstract molecular candidates into explicit, laboratory ready recipes. This shift from isolated molecular scoring to specification compliant blend design represents a central contribution of the work, aligning computational outputs with the requirements of experimental validation.

A further contribution lies in the system's emphasis on reproducibility. All optimisation runs are fully parameterised, deterministic, and accompanied by structured output artefacts that capture both results and execution context. This design enables exact regeneration of outputs and supports transparent analysis, reuse, and extension. In contrast to exploratory modelling workflows that prioritise novelty alone, Ignition Protocol treats reproducibility as a core methodological requirement.

The project also defines a clear pathway for experimental validation. Although physical testing was not conducted within the scope of this work, structured protocols for combustion fingerprinting and environmental screening were specified, and the computational pipeline produces the candidate lists and blend recipes required to execute these experiments reproducibly. This positions the system as a foundation for future mixed computational experimental studies rather than as a closed theoretical exercise.

Overall, Ignition Protocol demonstrates that effective early stage fuel design does not require exhaustive simulation or opaque machine learning models. Instead, a carefully structured pipeline that combines fast heuristics, deterministic optimisation, and explicit constraints can produce actionable and reproducible results. While further work is required to integrate high fidelity property models and experimental feedback, the current implementation establishes a robust baseline for computationally guided fuel formulation and iterative experimental exploration.

References

1] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2016. [Online]. Available: https://www.tensorflow.org

[2] A. G. Abdul Jameel et al., "Yield sooting index correlations for oxygenated fuels," Fuel, vol. 286, p. 119390, 2021.

[3] K. Aikawa et al., "Development of the Particulate Matter Index (PMI) for gasoline," SAE Technical Paper 2010-01-2115, 2010.

[4] J. Blank and K. Deb, "pymoo: Multi-objective optimization in Python," IEEE Access, vol. 8, pp. 89497–89509, 2020.

[5] M. Chen, K. Zhou, and D. Liu, "A novel AI-based combustion diagnostic technology for the identification of chemical source information via flame images," Combustion and Flame, vol. 260, p. 113208, 2024.

[6] Y. Chen et al., "Random-forest classification of flame images," Combustion and Flame, vol. 259, pp. 112–125, 2024.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, 2002.

[8] J. Degen, C. Wegscheid-Gerlach, A. Zaliani, and M. Rarey, "On the art of compiling and using 'drug-like' chemical fragment spaces," ChemMedChem, vol. 3, no. 10, pp. 1503–1507, 2008.

[9] EPA, "Reformulated gasoline and oxygenated fuels program: CO reduction rationale," U.S. Environmental Protection Agency, 1998.

[10] P. Ertl and A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions," J. Cheminformatics, vol. 1, p. 8, 2009.

[11] L. Fleitmann, A. M. Schweidtmann, M. Skiborowski, and A. Mitsos, "Molecular design of fuels for maximum spark-ignition engine efficiency," Energy & Fuels, vol. 37, no. 4, pp. 2213–2229, 2023.

[12] E. Galloni, "HCCI combustion control with fuel reactivity," Energy Convers. Manag., vol. 63, pp. 3–11, 2012.

[13] I. Glassman, R. A. Yetter, and N. Glumac, Combustion, 5th ed., Academic Press, 2015.

[14] S. R. Gollahalli, "Water–fuel emulsions: NOx/soot reduction," Prog. Energy Combust. Sci., vol. 9, no. 3, pp. 193–212, 1983.

[15] W. H. Green et al., "MolPAL: Multi-objective pool-based active learning for molecular discovery," Digital Discovery, vol. 3, pp. 375–390, 2024.

[16] R. Harris, "Automotive fuels: Octane quality and engine knock," SAE Technical Paper, 1990.

[17] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020.

[18] J. B. Heywood, Internal Combustion Engine Fundamentals, McGraw-Hill, 1988.

[19] C. Y. Hsuan, W. R. Jhang, W. J. Lin, and C. L. Su, "Water-in-oil emulsion for reduced NOx," Energies, vol. 12, no. 6, p. 1002, 2019.

[20] ICCT, "Global fuel sulfur limits and emission standards," Int. Council on Clean Transportation Report, 2016.

[21] Intel Corp., "12th Gen Intel® Core™ Processors Datasheet," 2022.

[22] H. Jääskeläinen, "Cetane number and diesel fuel quality," DieselNet Tech. Rep., 2007.

[23] W. Jin, R. Barzilay, and T. Jaakkola, "Junction Tree Variational Autoencoder," in Proc. ICML, pp. 2323–2332, 2018.

[24] G. T. Kalghatgi, "Fuel octane and auto-ignition in HCCI," SAE Tech. Paper 2001-01-3584, 2001.

[25] A. Karim and H. Mahmoud, "Quantifying flare combustion using MWIR FTIR," J. Environ. Monitoring, vol. 25, no. 3, pp. 199–211, 2024.

[26] G. A. Karim and M. Mahmoud, "Optical diagnostics for engine flames," Fuel, vol. 354, pp. 129–142, 2024.

[27] W. Kovarik, "Ethyl-leaded gasoline: Legacy of Midgley and Kettering," Public Understand. Sci., vol. 14, no. 1, pp. 3–39, 2005.

[28] M. Krenn et al., "SELFIES: Robust graph representation for chemistry," Mach. Learn.: Sci. Technol., vol. 1, no. 4, p. 045024, 2020.

[29] N. Kuzhagaliyeva et al., "AI-driven design of fuel mixtures," Commun. Chem., vol. 5, p. 76, 2022.

[30] G. Landrum, "Using SA_Score and NP_Score in RDKit," RDKit Blog, 2023.

[31] J. Leguy, Z. Zhang, and M. A. Kayala, "EvoMol: Flexible algorithm for molecular generation," J. Cheminformatics, vol. 12, no. 1, p. 55, 2020.

[32] L. C. Blum and J.-L. Reymond, "970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-11," J. Chem. Inf. Model., vol. 49, no. 11, pp. 2734–2745, 2009.

[33] T. G. Leone et al., "Effects of fuel octane and sensitivity on modern boosted SI engines," SAE Int. J. Fuels Lubr., vol. 7, no. 1, pp. 9–28, 2014.

[34] H. Li et al., "Filament-induced fluorescence for combustion intermediates," Appl. Phys. Lett., vol. 109, p. 104101, 2016.

[35] H. Li, W. Chu, H. Xu, Y. Cheng, S. L. Chin, and H. B. Sun, "Simultaneous identification of multi-combustion-intermediates of alkanol-air flames by femtosecond filament excitation," Sci. Rep., vol. 6, p. 27340, 2016.

[36] B. Liu, B. Ramsundar, P. Kawthekar, et al., "Retrosynthetic reaction prediction using neural sequence-to-sequence models," ACS Cent. Sci., vol. 3, no. 10, pp. 1103–1113, 2017.

[37] V. Makarevičienė, E. Sendžikienė, and P. Janulis, "Analysis of biological degradation and life cycle indicators of diesel fuel mixtures containing 10% biodiesel," Energies, vol. 14, no. 24, p. 8367, 2021.

[38] V. Makarevičienė et al., "Environmental impact of diesel–biodiesel blends," Renew. Energy, vol. 170, pp. 253–262, 2021.

[39] G. Mazzotta, J. Cha, and Y. J. Cha, "Monitoring ammonia–hydrogen combustion stability using flame chemiluminescence and deep learning," Fuel, vol. 350, p. 128731, 2025.

[40] W. McKinney, "Data structures for statistical computing in Python," in Proc. 9th Python in Science Conf., pp. 56–61, 2010.

[41] MolPAL Team, "Multi-objective active learning for Pareto fronts," Digital Discovery, vol. 3, pp. 227–245, 2024.

[42] D. Mazzotta, et al., "ML interpretation of NH_3–H_2 chemiluminescence," Combust. Sci. Technol., vol. 197, no. 6, pp. 945–965, 2025.

[43] H. Needleman, "Lead exposure and children's IQ," N. Engl. J. Med., vol. 322, no. 2, pp. 83–88, 1990.

[44] NIST Chemistry WebBook, "Thermochemical data & heats of vaporization," National Institute of Standards and Technology. [Online]. Available: https://webbook.nist.gov

[45] NVIDIA Corporation, "CUDA Toolkit Documentation," 2022. [Online]. Available: https://docs.nvidia.com/cuda/

[46] J. Pasqualino, D. Montané, and J. Salvadó, "Synergic effects of biodiesel in the biodegradability of fossil-derived fuels," Biomass Bioenergy, vol. 30, no. 10, pp. 874–879, 2006.

[47] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," Advances in Neural Information Processing Systems, vol. 32, pp. 8026–8037, 2019.

[48] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[49] H. Ritchie, "The global end of leaded gasoline," Our World in Data, 2022.

[50] S. Riniker and G. A. Landrum, "Better informed distance geometry: Using what we know to improve conformation generation," J. Chem. Inf. Model., vol. 55, no. 12, pp. 2562–2574, 2015.

[51] J. G. Rittig et al., "Graph ML for high-octane fuel discovery," AIChE J., vol. 69, no. 2, p. e17971, 2023.

[52] S. M. Sarathy and B. A. Eraqi, "Artificial intelligence for novel fuel design," Proc. Combust. Inst., vol. 40, no. 1, p. 105630, 2024.

[53] S. M. Sarathy and H. Eraqi, "AI in fuel development: A review," Prog. Energy Combust. Sci., vol. 94, p. 101061, 2024.

[54] M. H. S. Segler, M. Preuss, and M. P. Waller, "Planning chemical syntheses with deep neural networks and symbolic AI," Nature, vol. 555, pp. 604–610, 2018.

[55] R. Stone, Introduction to Internal Combustion Engines, 3rd ed., SAE International, 1999.

[56] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," Dr. Dobb's Journal, 2005.

[57] W. Tang, L. Zhang, Y. Wang, and Q. Zhou, "Design and construction of a synthetic bacterial consortium to enhance petroleum hydrocarbon degradation," J. Hazard. Mater., vol. 398, p. 122903, 2020.

[58] X. Tang et al., "Engineering microbes for fuel biodegradation," Biotechnol. Adv., vol. 42, p. 107582, 2020.

[59] UNEP, "Elimination of leaded petrol globally: Final report," United Nations Environment Programme, 2021.

[60] U.S. DOE Alternative Fuels Data Center, "Fuel properties and energy content," U.S. Department of Energy, 2020.

[61] V. Venkatasubramanian, K. Chan, and J. M. Caruthers, "Computer-aided molecular design using genetic algorithms," Comput. Chem. Eng., vol. 18, no. 9, pp. 833–844, 1994.

[62] V. Venkatasubramanian, A. Sundaram, P. Ghosh, and J. M. Caruthers, "Design of fuel additives using neural networks and evolutionary algorithms," AIChE J., vol. 48, no. 5, pp. 986–996, 2002.

[63] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nat. Methods, vol. 17, pp. 261–272, 2020.

[64] Wikipedia, "Evolutionary computation," Wikipedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Evolutionary_computation

[65] K. Yang et al., "Analyzing learned molecular representations for property prediction," J. Chem. Inf. Model., vol. 59, no. 8, pp. 3370–3388, 2019.

[66] K. K. Yalamanchi et al., "Generative deep learning framework for inverse design of fuels," arXiv preprint, arXiv:2504.12075, 2025.

[67] P. Yalamanchi et al., "Generative models targeting high-octane molecules," Fuel, vol. 356, p. 129495, 2025.

[68] C. Yin et al., "Spectral fingerprinting of liquid fuel flames," Combust. Sci. Technol., vol. 190, no. 4, pp. 621–639, 2018.

[69] F. Yin, X. Zhou, and L. Chen, "Biogas combustion: chemiluminescence fingerprint," J. Anal. Appl. Pyrolysis, 2018.

[70] S. Genheden et al., "AiZynthFinder: A fast, robust and flexible open-source software for retrosynthetic planning," Journal of Cheminformatics, vol. 12, p. 70, 2020.

[71] A. Nigam, R. Pollice, M. Krenn, G. dos Passos Gomes, and A. Aspuru-Guzik, "Beyond generative models: Superfast traversal, optimization, novelty, exploration and discovery (STONED) algorithm for molecules using SELFIES," Chemical Science, vol. 12, pp. 7079–7090, 2021.

[72] J. Fleitmann, D. Reuss, and U. Maas, "Evolutionary optimization of fuel molecules for SI engines," Fuel, vol. 345, p. 128180, 2023.

[73] J. Leguy et al., "Evolutionary algorithms for inverse molecular design," Digital Discovery, vol. 1, pp. 48–61, 2020.

[74] RDKit: Open-source cheminformatics. http://www.rdkit.org