Lightbeam technical overview

Abstract

Data integration and software usability have long been challenges for every analytics vendor, with lengthy change management managed by large teams of contractors ending in surprisingly low user adoption being the norm in the enterprise data industry. We are now entering an age — the era of bigger data, more advanced machine learning techniques, and increasingly complex networks of IoT devices producing ever-larger real-time data streams — where it is of paramount strategic importance to finally solve these challenges in a highly general and repeatable way.

Lightbeam is a new cognitive computing product that solves these problems and provides an intuitive English-based interface to data integration, visualization, statistics, predictive analytics, and mathematical model creation. This paper shall describe the design philosophy behind Lightbeam's architecture, illustrate the intended scope of questions that Lightbeam will be equipped to answer for its users, and provide a high-level overview of the systems that work together to answer those questions.

Architectural approach

The following engineering insights form the core of the Lightbeam ethos. If there are good reasons that nobody else has produced Lightbeam, they lie in this section.

Focus on business value

Lightbeam's first engineering value is that architecture at Lightbeam is driven by solving business problems and making usable software, *not* by making things easy for our internal developers. To that end, Lightbeam's architecture has been designed from the ground up to *work for the user*, and to avoid forcing the user to work for Lightbeam.

The most salient example is, of course, our unique English-based query system, whose intuitiveness avoids requiring the user to sit through training courses or click dropdown menus, but our business focus extends much further. A further example is, in fact, how the data integration gets carried out. Under the hood, it behaves much like SQL, with one additional item: we have found that users of traditional BI software have difficulty with joins, and so Lightbeam *deduces* the joins using our graph-based data architecture instead of requiring the user to explicitly specify them.

Another example is our data integration system itself. Vendors usually design software with an API, so that proprietary customer databases can be hooked up to the software with some vendor contractor involvement. In addition to that, Lightbeam has elected to go a step further and design a system that automatically integrates customer schemas, avoiding the need for contractors to create and import a detailed schema.

Correctness

With an eye toward the high-stakes world of our clientele, our second core engineering value is correctness — the mathematically guaranteed correctness of our programs. This manifests itself in Lightbeam's architecture both in its own structure and in the technologies we've chosen as the foundation of Lightbeam.

The most significant guarantee of correctness in Lightbeam is our use of Haskell as the programming language for both our frontend and backend. Compared to the languages in vogue in analytics, like Python, R, and Matlab, and even compared to other statically typed languages like Java and C++, Haskell biases much more toward compile-time correctness checking. In plainer language, you find out a Python program is wrong when you run it and get an incorrect result, but Haskell warns you before you ever run the program. For the highly heterogeneous and complex workloads Lightbeam is intended to run, this is not just a plus, but a necessity.

Going one step further, we have implemented a type system taking inspiration from Haskell's for Lightbeam's own English queries. This has both near-term and long-term implications:

- In the near term, unlike in WolframAlpha, our goal is to ensure that Lightbeam users will never have the experience of typing in a query and finding out it doesn't work but not knowing why. This is due to our type system and autocorrect features, which ensure that only valid questions are allowed to be typed, and that if the user enters an invalid command, they are presented with a clear reason for its failure, in plain English.
- In the long term, we aim to solve what we regard to be a major hidden problem with data analytics: that it is based on the faulty foundation of dynamic typing provided by the major languages used in data science. This undermines repeatability and the trustworthiness of results produced from Python and R programs. By eliminating these problems, Lightbeam aims to simultaneously be safer to use and more intuitive, and ultimately aims to replace all unsafe systems with provably flawless ones.

Performance

Performance is also a core value of Lightbeam, and Lightbeam's use of Haskell further pays off in its performance. Haskell, among programming languages, is exceptional in being both high-level (concise) and performant. For example, Haskell boasts the most performant web server in the world and Facebook uses Haskell to run its spam filter engine, which deals with some of the highest data volumes in the world.

How it works

Notebook interface

Our notebook app is the starting point for any question asked of Lightbeam. Lightbeam's notebook interface is a combination of the best features of Google Docs and IPython. Specifically, it has the advanced technical editing capabilities of IPython (autocomplete and in place error checking) as well as the collaborative features of Google Docs (simultaneous editing and sharing of documents via links). Indeed, being an interactive notebook environment, Lightbeam can actually be used as a more powerful and intuitive dashboarding and app creation tool than Shiny and RStudio.

Questions that you can ask

Lightbeam allows users to query, analyze, and visualize their data, as well as create simulations. Our standard library of functionality includes:

- Data integration provided by Lightbeam's Prism engine
- Machine learning and statistics
    - Clustering
    - Factor analysis and PCA
    - Regression
    - Classification
    - Support vector machines, neural networks
    - Time series analytics (ARIMA, anomaly detection, etc.)
    - Network and graph analytics
- Mathematics
    - A full CAS system, including everything from arithmetic to PDEs and tensor calculus
- Computational simulations
    - Fluid systems
    - Statics and solid body dynamics
    - Reactor modeling

- An API that allows you to easily add new functionality

Linguistic processing system

We use state-of-the-art nondeterministic parser technology. This sophisticated language processing engine enables Lightbeam to discern the user's intent. Lightbeam then uses the plain English query to generate code in its intermediate language, *Photon*, which is a full-featured, Turing-complete functional programming language.

Photon: the computational bedrock

Lightbeam's entire extensive library is implemented in Photon, its proprietary programming language. Photon takes its inspiration from the most powerful features of other programming languages and is distinguished from typical programming languages in several ways:

- Its evaluation engine is a term rewriting calculus capable of running computer algebra system (CAS) operations.
- Unlike other CAS languages, Photon is a pure functional programming language, which ensures correctness.
- Photon's unique use of typeclasses, a concept borrowed from the Haskell type system, allows it to aid the linguistic processing system in inferring the user's intent.
  - For example, in a situation where Lightbeam is presented with the query, "*rerun that model on all similar bioreactors*," Lightbeam *extrapolates* the model and deduces how it can apply it to several bioreactors, not just one, and *infers* that it should do so *in parallel* over all of its compute nodes.
  - Another example is that Lightbeam can trivially implement probabilistic programming and Monte Carlo simulations on top of other queries.
- Photon is also a logic programming language, meaning that it can logically deduce general statements from other general statements.

All of these features significantly add to the overall deductive intelligence of Lightbeam.

Prism: the data integration engine

Before Lightbeam can fully answer a question from a user, the underlying data must be brought together from the vast set of data sources behind a modern enterprise. This is where Prism, our advanced data integration engine, comes in. Prism is based on cutting-edge work at a leading pharmaceutical company, and based on novel and recent academic research in the data integration field that has mostly yet to be put to business use.

During its initial setup phase, Prism uses a heuristics-based automatic schema detection engine to learn your schema instantly — instead of expensive contractors manually assembling it over a long period of time! Similar to how Google automatically crawls the web, Prism automatically crawls your data sources, which can include sources like OSISoft PI Assets and Events, ISA 88 Recipes, and Symyx/Accelrys ELN databases, to put together a unified picture of how they work together.

When a query comes in, instead of translating it into SQL, Prism turns it into a graph database query. Graph queries are equally powerful as SQL joins, but much easier to optimize and parallelize. They also take away some of the burden from end users: instead of writing artificially constructed phrases like *reactors where the building ID is B5*, you can just write *reactors in B5*. This is similar to how Facebook Graph Search allows non-technical users to ask questions like *who are the friends of my friends who like the Beatles?*

Our graph database then delegates execution of more computationally intensive time series query fragments to Lightbeam's built-in column store; otherwise, it assembles the data you request and hands it to Prism, which returns the answer back to you.

Conclusion

Lightbeam is built from a unique assembly of cutting-edge technologies and with an awareness that the ultimate purpose of technology is to solve human problems, not computer problems. We dare to believe that with a dose of intelligence and with a dose of humanity, we can make analytics more advanced than mere pivot tables, more intuitive than learning to code in Python, and more trustworthy than today's folders chock-full of scattershot scatterplots, all at the same time. We think the field of data science is long overdue for this kind of thinking, and we hope you'll join us in our revolution.