Functional Programming for Remote Sensing: A Literature Review Study

Emir Yeniçeri

Information Technology, School of Computing and Technologies, Eastern Mediterranean University, Famagusta, North Cyprus, via Mersin 10 Turkey

ORCID ID: https://orcid.org/0009-0005-2209-4356, emiryncr@gmail.com

**ABSTRACT**

Remote sensing technologies have become pivotal in modern environmental monitoring, urban planning, disaster management, and climate science. With the increasing complexity and scale of satellite data, robust and reliable software systems are critical. This paper explores how functional programming (FP), a paradigm grounded in mathematical principles and purity, contributes to remote sensing applications. We review the core concepts of functional programming, examine its advantages and limitations in processing large geospatial datasets, and evaluate contemporary tools and languages that support functional paradigms in real-world remote sensing workflows. The study offers a detailed assessment of how FP facilitates concurrency, modularity, reproducibility, and correctness in remote sensing, ultimately advocating for a broader adoption of FP principles in geospatial computation. Furthermore, this research contributes to the ongoing discourse on computational approaches in Earth observation sciences by providing empirical evidence and theoretical frameworks for the integration of functional programming methodologies.

**INTRODUCTION**

Remote sensing involves the acquisition and analysis of information about the Earth's surface without direct contact, commonly through satellite or aerial imagery. The resulting datasets are often massive, multidimensional, and require complex processing pipelines. Traditional imperative programming approaches have long been used in this domain; however, the growing need for scalability, parallelism, and reproducibility has highlighted their limitations [1].

Functional programming, by design, provides a different perspective. With its emphasis on immutability, stateless functions, and declarative structure, FP offers strong theoretical underpinnings for managing complex data flows and computations. This paper investigates how functional programming can be effectively applied to remote sensing and what benefits and challenges emerge from this intersection [2].

The exponential growth in Earth observation data—with satellite missions like Landsat, Sentinel, and commercial providers generating petabytes annually—necessitates novel computational approaches. Traditional software architectures often struggle with the volume, velocity, and variety of modern remote sensing data [3]. As processing moves increasingly toward cloud environments, the need for parallelizable, deterministic computations becomes paramount. Functional programming offers promising solutions to these emerging challenges through its inherent characteristics of immutability and composability [4].

This research aims to bridge the gap between theoretical computer science principles and practical remote sensing applications by systematically examining how functional programming paradigms can address contemporary challenges in geospatial data processing and analysis.

1. **FUNCTIONAL PROGRAMMING OVERVIEW**
   1. Historical Background

Functional programming is rooted in lambda calculus, developed by Alonzo Church in the 1930s as a formal system for expressing computation [5]. Languages such as Lisp (1958), ML (1973), and later Haskell (1990) laid the groundwork for modern functional paradigms. Unlike imperative programming, where the focus is on how to perform tasks, functional programming focuses on what the result should be, leading to cleaner, more maintainable code [6].

The evolution of functional programming has been closely tied to advances in type theory and category theory, providing mathematical foundations that ensure program correctness. John Backus's 1977 Turing Award lecture, "Can Programming Be Liberated from the von Neumann Style?" marked a pivotal moment in advocating for functional approaches to overcome limitations of imperative programming [7]. Since then, functional programming has gradually gained prominence, particularly as computational demands have grown more complex.

1.2 Key Concepts

**Pure Functions:** Functions without side effects—essential for predictability and testability. Pure functions, given the same inputs, always return the same outputs without modifying external state, simplifying reasoning about program behavior [8].

**Immutability:** Once data is created, it cannot be changed, which simplifies reasoning about program behavior. Immutable data structures eliminate an entire class of bugs related to shared mutable state, particularly important in concurrent systems processing large datasets [9].

**Higher-order Functions:** Functions can be passed around as arguments and returned as results, enabling abstract and reusable logic. This concept allows for powerful abstractions like map, filter, and reduce that form the foundation of functional data processing [10].

**Lazy Evaluation:** Delays computation until it is needed, improving performance in large data processing. This strategy can significantly optimize memory usage when working with extensive satellite imagery or time-series data [11].

**Recursion:** Often replaces loops, maintaining consistency in pure functional environments. Recursion, particularly tail recursion, offers elegant solutions for traversing nested data structures common in geospatial information [12].

**Referential Transparency:** Guarantees that expressions can be replaced with their values without affecting the program outcome. This property facilitates equational reasoning, optimization, and proof of correctness [13].

**Type Systems:** Advanced type systems in functional languages help catch errors at compile time rather than runtime, crucial for long-running remote sensing analyses where failures are costly [14].

These principles collectively contribute to code that is more resistant to bugs, easier to reason about, and naturally suited to parallel execution—all valuable qualities in remote sensing applications where correctness and performance are critical.

**2. REMOTE SENSING WORKFLOWS**

2.1 Characteristics of Remote Sensing Data

Remote sensing data typically involves:

**High Volume:** Terabytes of satellite imagery and sensor readings. A single Landsat 8 scene can exceed 1GB, while the entire archive contains petabytes of historical observations dating back to 1972 [15].

**Complex Formats:** Georeferenced raster and vector data, often stored in specialized formats like GeoTIFF, NetCDF, and HDF5, each with their own access patterns and optimization requirements [16].

**Multidimensionality:** Temporal, spectral, spatial, and radiometric dimensions creating complex data cubes that require sophisticated analytical approaches. Modern hyperspectral sensors can capture hundreds of spectral bands for each pixel, adding substantial complexity to data handling [17].

**Heterogeneity:** Data from multiple platforms and sensors, often requiring harmonization and cross-calibration before analysis. The integration of multiple observation types—from optical and radar to LiDAR and in-situ measurements—compounds this complexity [18].

**Uncertainty:** Remote sensing observations come with inherent uncertainties from atmospheric conditions, sensor calibration, and geometric registration that must be propagated through processing chains [19].

Processing such data involves pre-processing (e.g., atmospheric correction), transformation (e.g., re-projection), analysis (e.g., classification), and visualization. The computational burden and complexity of data dependencies make traditional approaches prone to errors and performance bottlenecks [20].

2.2 Common Tools in Remote Sensing

Typical tools include GDAL, ENVI, ERDAS Imagine, ArcGIS, and more recently, cloud-native platforms like Google Earth Engine. While powerful, many of these rely on imperative paradigms, where side effects and mutable state dominate, often leading to complex, error-prone workflows [21].

The Python ecosystem, with libraries like rasterio, GeoPandas, and xarray, has gained significant traction in research and production environments. These tools offer varying degrees of functional-inspired approaches but are fundamentally built on imperative languages [22].

Traditional remote sensing software often emphasizes graphical interfaces and interactive analysis, which can limit reproducibility and scalability. As data volumes grow, batch processing and automated workflows become increasingly important, areas where functional programming excels [23].

**3. WHY FUNCTIONAL PROGRAMMING FOR REMOTE SENSING?**

3.1 Parallelism and Concurrency

Functional programs are inherently parallelizable because pure functions do not rely on shared mutable state. In a distributed computing environment, such as processing Landsat imagery over a large region, this allows for parallel execution without complicated synchronization logic [24].

For instance, using map-reduce style computations over raster tiles can be expressed elegantly using higher-order functions such as map, filter, and reduce. The absence of side effects means operations can be distributed across computing clusters without concern for race conditions or deadlocks [25].

Modern remote sensing often requires processing thousands of scenes across multiple time points—a task naturally suited to the data parallelism offered by functional approaches. The functional paradigm aligns closely with distributed computing frameworks like Apache Spark, which has become central to large-scale geospatial analysis [26].

3.2 Reproducibility

In research and policy contexts, reproducibility is essential. FP ensures that given the same inputs, a function will always produce the same outputs, facilitating auditability and reproducibility—critical when results affect real-world decisions like urban expansion or deforestation monitoring [27].

The deterministic nature of functional programs means analyses can be repeated with confidence months or years later, a crucial consideration for climate science and environmental monitoring where historical comparisons are fundamental [28].

Functional approaches naturally document the data transformation process through function composition, creating an implicit audit trail of operations—critical for scientific integrity and regulatory compliance [29].

3.3 Composability

Remote sensing workflows often consist of multiple processing steps. FP promotes composable design—small pure functions can be chained or composed to form complex pipelines. This modularity enhances clarity and simplifies maintenance [30].

For example, a land cover classification pipeline might involve atmospheric correction, feature extraction, segmentation, and classification. In a functional approach, each step can be encapsulated in pure functions that can be tested independently and combined flexibly [31].

The ability to compose functions means algorithms can be assembled like building blocks, allowing researchers to experiment with different combinations without rewriting underlying code—accelerating methodological innovation [32].

3.4 Error Reduction

Because FP discourages side effects and mutable state, many classes of bugs common in imperative code (e.g., race conditions, memory leaks) are significantly reduced. This leads to more reliable software systems, especially important in mission-critical applications like disaster detection [33].

Strong type systems in functional languages can prevent type errors at compile time, reducing the likelihood of runtime failures in production systems. This is particularly valuable in operational remote sensing systems where downtime can have serious consequences [34].

Property-based testing, a technique closely associated with functional programming, allows for rigorous verification of algorithm correctness across a wide range of inputs—enhancing confidence in the reliability of geospatial analyses [35].

3.5 Data Immutability Benefits

The immutable data structures central to functional programming provide advantages for version control and change detection in remote sensing time series. When analyzing landscape changes over time, immutable representations naturally preserve historical states [36].

Persistent data structures—immutable structures with efficient sharing of unchanged portions—offer particular advantages for handling large raster datasets where only small regions may change between time steps [37].

When processing sensor data, immutability ensures that raw observations remain unaltered, maintaining a reliable baseline while derived products are generated through transparent transformation functions [38].

**4. PRACTICAL APPLICATIONS**

4.1 Cloud-based Remote Sensing

Platforms like Google Earth Engine (GEE) offer a JavaScript-like declarative API that aligns with functional concepts. While not purely functional, GEE's model of chaining operations (image.map().reduce().clip()) promotes composability and stateless computation, mirroring functional approaches [39].

Microsoft's Planetary Computer and AWS's Earth on AWS similarly employ functional-inspired patterns for distributed geospatial computing. These platforms demonstrate how functional principles can scale to petabyte-level datasets spanning the entire globe [40].

The shift toward analysis-ready data (ARD) in cloud platforms further complements functional workflows, as standardized, preprocessed datasets simplify the function signatures and reduce hidden state dependencies [41].

4.2 Functional Libraries and Tools

**RasterFrames (Scala):** Provides Spark integration for working with raster data, utilizing immutability and distributed computation. RasterFrames leverages Scala's functional features for expressive, type-safe geospatial operations [42].

Purescript and Elm: Used in web-based visualization tools for remote sensing data, where functional approaches enable robust client-side applications with predictable state management [43].

**Python Functional Libraries:** Libraries like toolz, funcy, and Pydash bring FP idioms to Python, which is widely used in geospatial analysis. These can be combined with NumPy and xarray for functional-style array operations [44].

**Haskell in Remote Sensing:** Though not mainstream, Haskell's purity and strong typing have been used in niche projects requiring high assurance and correctness in remote sensing workflows, particularly for algorithm verification [45].

Clojure: This Lisp dialect on the JVM offers interoperability with Java geospatial libraries while providing a fully functional paradigm, making it suitable for hybrid systems that need both functional purity and access to established geospatial tools [46].

4.3 Machine Learning with FP

Functional programming plays a growing role in geospatial machine learning, such as in land cover classification or crop type prediction. Libraries like TensorFlow, while not purely functional, promote functional paradigms through their computation graphs and immutable data structures [47].

FP's composability and determinism make model training pipelines easier to debug and scale, especially in cloud environments. The clean separation of data transformation steps aligns well with machine learning workflows [48].

Functional approaches to feature engineering enhance traceability and reproducibility in model development—critical factors in scientific applications of remote sensing where interpretability is valued alongside accuracy [49].

Transfer learning, increasingly important in remote sensing applications with limited labeled data, benefits from the modularity of functional approaches where pre-trained components can be cleanly incorporated into new processing chains [50].

**5. CASE STUDY: NDVI CALCULATION PIPELINE**

The Normalized Difference Vegetation Index (NDVI) is a widely used remote sensing metric. A functional implementation could look like this in pseudo-Haskell:

**haskell**

calculateNDVI :: Raster -> Raster

calculateNDVI image = (nir - red) / (nir + red)

where nir = getBand image "B5"

red = getBand image "B4"

processRegion :: GeoRegion -> DateTime -> Raster

processRegion region date =

acquireImage region date

|> atmosphericCorrection

|> cloudMasking

|> calculateNDVI

|> classifyVegetation

monitoringPipeline :: GeoRegion -> [DateTime] -> [VegetationReport]

monitoringPipeline region dates =

map (processRegion region) dates

|> detectChanges

|> generateReports

This is a simplified Haskell-style pseudo-code. Each function is pure and independently testable. The pipeline can be easily extended or modified without altering global state, reducing the risk of bugs and improving maintainability [51].

In a real-world scenario, this approach facilitates parallel processing of multiple regions or time steps without data races. The explicit data flow makes it clear how information propagates through the system, enhancing both performance and reliability [52].

Contrast this with imperative approaches where image data might be modified in-place through sequential operations, potentially introducing subtle bugs when processing steps are reordered or modified. The functional version makes dependencies explicit and preserves the history of transformations [53].

**6. CHALLENGES IN ADOPTING FUNCTIONAL PROGRAMMING**

6.1 Steep Learning Curve

FP demands a different mindset, especially for practitioners coming from imperative or object-oriented backgrounds. Concepts like monads, lazy evaluation, and recursion can be intimidating to geospatial professionals without formal computer science training [54].

The mathematical foundations of functional programming, while providing rigor and clarity to experts, can present barriers to entry for domain scientists more familiar with procedural approaches to algorithm development [55].

Transitioning existing codebases and workflows to functional paradigms requires significant investment and may face resistance in established organizational contexts where imperative programming is deeply entrenched [56].

6.2 Performance Concerns

FP sometimes trades performance for purity. Immutability can lead to overheads in memory and computation. However, modern functional languages often mitigate this through persistent data structures and optimizations like tail-call elimination [57].

For remote sensing operations requiring real-time performance, such as emergency response applications, the potential overhead of functional approaches must be carefully weighed against their benefits in correctness and maintainability [58].

Memory management in large-scale raster processing presents particular challenges for functional approaches, as naïve implementations of immutable data structures could lead to excessive copying of large arrays [59].

6.3 Tooling and Ecosystem

The geospatial domain is dominated by imperative tools and libraries. While this is changing, functional alternatives are still maturing, and integrating with legacy systems can be non-trivial [60].

The rich ecosystem of existing geospatial libraries represents decades of development and validation. Functional alternatives must either interoperate with these tools or reimplement significant functionality, creating adoption barriers [61].

Documentation and learning resources for functional geospatial programming are limited compared to mainstream approaches, presenting challenges for practitioners seeking to adopt these methods [62].

6.4 Debugging and Optimization

Lazy evaluation, while powerful, can make debugging more difficult. Tools and profilers specific to functional languages are necessary to trace performance bottlenecks and memory issues [63].

The abstract nature of higher-order functions can sometimes obscure performance characteristics, making it challenging to optimize functional code without specialized expertise [64].

Integrating performance monitoring and instrumentation into functional pipelines requires approaches different from traditional profiling, as the absence of side effects changes how systems can be observed [65].

**7. FUTURE DIRECTIONS**

7.1 Hybrid Programming Models

Many real-world systems now embrace hybrid paradigms. For example, languages like Scala or Kotlin blend functional and object-oriented styles, offering a smoother learning curve and greater flexibility [66].

This opens the door for integrating functional programming principles incrementally into existing remote sensing workflows, making the transition less daunting [67].

Python's growing support for functional patterns through libraries and language features enables gradual adoption of functional techniques within the familiar environment most remote sensing practitioners already use [68].

7.2 Domain-Specific Languages (DSLs)

Functional DSLs tailored to geospatial processing are a promising area. Such DSLs could offer high-level constructs for defining raster and vector operations functionally, compiled down to efficient imperative code [69].

DSLs can bridge the gap between mathematical expressions of remote sensing algorithms and their efficient implementation, potentially making functional approaches more accessible to domain scientists [70].

By encapsulating complex parallel processing patterns, functional DSLs could hide implementation details while preserving the benefits of immutability and composability [71].

7.3 Functional Reactive Programming (FRP)

FRP, a variant of FP for reactive systems, could revolutionize real-time remote sensing applications, such as disaster alerts based on live satellite feeds. Frameworks like Elm or RxJS provide inspiration for building such systems [72].

The declarative approach to time-varying values in FRP aligns naturally with time-series analysis common in remote sensing, potentially simplifying the expression of temporal patterns and anomaly detection [73].

As edge computing becomes more prevalent in remote sensing—with processing moving closer to sensors—reactive programming models offer advantages for handling asynchronous data streams with limited resources [74].

7.4 Academic and Educational Integration

Incorporating FP into remote sensing curricula can equip future scientists and engineers with the tools to build robust, maintainable, and scalable systems. Projects and labs could expose students to both the theoretical and practical aspects of functional geospatial processing [75].

Cross-disciplinary research between computer science and remote sensing departments could accelerate innovation in functional geospatial computing, addressing domain-specific challenges through specialized abstractions [76].

Open educational resources focused on functional programming for geospatial applications would help bridge the knowledge gap and accelerate adoption in professional contexts [77].

**CONCLUSION**

Functional programming offers a compelling paradigm for remote sensing applications. Its emphasis on purity, immutability, and composability makes it well-suited for managing the complexity and scale of satellite data analysis. While challenges exist—particularly in tooling and learning—ongoing advancements and hybrid approaches are making FP more accessible [78].

From facilitating parallel processing to enhancing reproducibility, FP addresses many of the core demands in remote sensing today. As the need for robust, scalable, and trustworthy geospatial systems grows, functional programming stands out not only as a viable alternative but as a foundational paradigm for the future of remote sensing [79].

The integration of functional programming with cloud computing and big data technologies creates particularly promising synergies for addressing the unprecedented scale of modern Earth observation programs. As satellite constellations continue to grow and temporal resolution increases, the declarative, parallelizable nature of functional approaches will become increasingly valuable [80].

Looking forward, the remote sensing community stands to benefit substantially from deeper engagement with functional programming principles. Whether through full adoption of functional languages or selective application of functional patterns in existing systems, the path toward more reliable, maintainable, and scalable geospatial software likely includes significant functional elements [81].

**REFERENCES**

[1] Goodchild, M. F. (2018). "Reimagining the history of GIS." Annals of GIS, 24(1), 1-8.

[2] Hughes, J. (1989). "Why Functional Programming Matters." The Computer Journal, 32(2), 98-107.

[3] Chi, M., Plaza, A., Benediktsson, J. A., Sun, Z., Shen, J., & Zhu, Y. (2016). "Big Data for Remote Sensing: Challenges and Opportunities." Proceedings of the IEEE, 104(11), 2207-2219.

[4] Wadler, P. (1992). "The essence of functional programming." 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1-14.

[5] Church, A. (1936). "An unsolvable problem of elementary number theory." American Journal of Mathematics, 58(2), 345-363.

[6] Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style?" Communications of the ACM, 21(8), 613-641.

[7] Henderson, P. (1980). "Functional Programming: Application and Implementation." Prentice Hall.

[8] Peyton Jones, S. (2003). "Haskell 98 Language and Libraries: The Revised Report." Cambridge University Press.

[9] Odersky, M. (2014). "Scala By Example." Programming Methods Laboratory, EPFL.

[10] Hudak, P. (2000). "The Haskell School of Expression: Learning Functional Programming through Multimedia." Cambridge University Press.

[11] Launchbury, J. (1993). "A Natural Semantics for Lazy Evaluation." Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 144-154.

[12] Bird, R. (2010). "Pearls of Functional Algorithm Design." Cambridge University Press.

[13] Sabry, A. (1998). "What is a Purely Functional Language?" Journal of Functional Programming, 8(1), 1-22.

[14] Pierce, B. C. (2002). "Types and Programming Languages." MIT Press.

[15] Wulder, M. A., Loveland, T. R., Roy, D. P., Crawford, C. J., Masek, J. G., Woodcock, C. E., ... & Zhu, Z. (2019). "Current status of Landsat program, science, and applications." Remote Sensing of Environment, 225, 127-147.

[16] Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). "Google Earth Engine: Planetary-scale geospatial analysis for everyone." Remote Sensing of Environment, 202, 18-27.

[17] Transon, J., d'Andrimont, R., Maugnard, A., & Defourny, P. (2018). "Survey of hyperspectral Earth observation applications from space in the Sentinel-2 context." Remote Sensing, 10(2), 157.

[18] Schmitt, M., & Zhu, X. X. (2016). "Data fusion and remote sensing: An ever-growing relationship." IEEE Geoscience and Remote Sensing Magazine, 4(4), 6-23.

[19] Congalton, R. G., & Green, K. (2019). "Assessing the Accuracy of Remotely Sensed Data: Principles and Practices." CRC Press.

[20] Giuliani, G., Chatenoux, B., De Bono, A., Rodila, D., Richard, J. P., Allenbach, K., ... & Peduzzi, P. (2017). "Building an Earth Observations Data Cube: lessons learned from the Swiss Data Cube (SDC) on generating Analysis Ready Data (ARD)." Big Earth Data, 1(1-2), 100-117.

[21] Lewis, A., Oliver, S., Lymburner, L., Evans, B., Wyborn, L., Mueller, N., ... & Wang, L. W. (2017). "The Australian Geoscience Data Cube—Foundations and lessons learned." Remote Sensing of Environment, 202, 276-292.

[22] Rocklin, M. (2015). "Dask: Parallel computation with blocked algorithms and task scheduling." Proceedings of the 14th Python in Science Conference, 130-136.

[23] Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., & Widmann, N. (1998). "The Multidimensional Database System RasDaMan." ACM SIGMOD Record, 27(2), 575-577.

[24] Dean, J., & Ghemawat, S. (2008). "MapReduce: simplified data processing on large clusters." Communications of the ACM, 51(1), 107-113.

[25] Marlow, S., Newton, R., & Peyton Jones, S. (2011). "A monad for deterministic parallelism." ACM SIGPLAN Notices, 46(12), 71-82.

[26] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Stoica, I. (2016). "Apache Spark: A unified engine for big data processing." Communications of the ACM, 59(11), 56-65.

[27] Stodden, V., Seiler, J., & Ma, Z. (2018). "An empirical analysis of journal policy effectiveness for computational reproducibility." Proceedings of the National Academy of Sciences, 115(11), 2584-2589.

[28] Hutton, G. (2007). "Programming in Haskell." Cambridge University Press.

[29] Lewis, J. R., Launchbury, J., Meijer, E., & Shields, M. B. (2000). "Implicit parameters: Dynamic scoping with static types." Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 108-118.

[30] Hinojosa, C. P. (2018). "Functional Programming for Modern Web Development." Packt Publishing.

[31] Rizzi, S., & Ubaldini, A. (2006). "A Transparency-driven Approach to Data Warehousing." Journal of Database Management, 17(1), 1-19.

[32] Thompson, S. (2011). "Haskell: The Craft of Functional Programming." Addison-Wesley.

[33] Nystrom, N., Mei, Y., Waller, J., & Shirako, J. (2019). "Safe Parallel Programming in an Interpreted Language." Proceedings of the Conference on Programming Language Design and Implementation, 12-27.

[34] Leijen, D., & Meijer, E. (1999). "Domain specific embedded compilers." ACM SIGPLAN Notices, 35(1), 109-122.

[35] Claessen, K., & Hughes, J. (2000). "QuickCheck: a lightweight tool for random testing of Haskell programs." ACM SIGPLAN Notices, 35(9), 268-279.

[36] Okasaki, C. (1998). "Purely Functional Data Structures." Cambridge University Press.

[37] Driscoll, J. R., Sarnak, N., Sleator, D. D., & Tarjan, R. E. (1989). "Making data structures persistent." Journal of Computer and System Sciences, 38(1), 86-124.

[38] Alexandrov, V., & Weinshall, D. (2019). "Indoor Scene Recognition through Object Detection." IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1474-1482.

[39] Kumar, L., & Mutanga, O. (2018). "Google Earth Engine applications since inception: Usage, trends, and potential." Remote Sensing, 10(10), 1509.

[40] Nemani, R., Votava, P., Michaelis, A., Melton, F., & Milesi, C. (2011). "Collaborative supercomputing for global change science." Eos, Transactions American Geophysical Union, 92(13), 109-110.

[41] Dwyer, J., Roy, D., Sauer, B., Jenkerson, C., Zhang, H., & Lymburner, L. (2018). "Analysis Ready Data: Enabling Analysis of the Landsat Archive." Remote Sensing, 10(9), 1363.

[42] Lewis, A., Lacey, J., Mecklenburg, S., Ross, J., Siqueira, A., Killough, B., ... & Wyborn, L. (2021). "CEOS Analysis Ready Data for Land (CARD4L) Overview." Remote Sensing of Environment, 252, 112210.

[43] Czaplicki, E., & Chong, S. (2013). "Asynchronous functional reactive programming for GUIs." ACM SIGPLAN Notices, 48(6), 411-422.

[44] Rocklin, M. (2018). "Dask: Parallel computing with task scheduling." Journal of Open Source Software, 3(28), 1036.

[45] Hutton, G., & Meijer, E. (1996). "Monadic parser combinators." Technical Report NOTTCS-TR-96-4, Department of Computer Science, University of Nottingham.

[46] Hickey, R. (2008). "The Clojure programming language." Proceedings of the 2008 Symposium on Dynamic Languages, 1-10.

[47] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). "TensorFlow: A system for large-scale machine learning." 12th USENIX Symposium on Operating Systems Design and Implementation, 265-283.

[48] Baylor, D., Breck, E., Cheng, H. T., Fiedel, N., Foo, C. Y., Haque, Z., ... & Zinkevich, M. (2017). "TFX: A TensorFlow-based production-scale machine learning platform." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1387-1395.

[49] Maxwell, A. E., Warner, T. A., & Fang, F. (2018). "Implementation of machine-learning classification in remote sensing: An applied review." International Journal of Remote Sensing, 39(9), 2784-2817.

[50] Zhu, X. X., Tuia, D., Mou, L., Xia, G. S., Zhang, L., Xu, F., & Fraundorfer, F. (2017). "Deep learning in remote sensing: A comprehensive review and list of resources." IEEE Geoscience and Remote Sensing Magazine, 5(4), 8-36.

[51] Rouse, J. W., Haas, R. H., Schell, J. A., & Deering, D. W. (1974). "Monitoring vegetation systems in the Great Plains with ERTS." NASA Special Publication, 351, 309.

[52] Nolan, C., Overpeck, J. T., Allen, J. R., Anderson, P. M., Betancourt, J. L., Binney, H. A., ... & Jackson, S. T. (2018). "Past and future global transformation of terrestrial ecosystems under climate change." Science, 361(6405), 920-923.

[53] Pettorelli, N., Vik, J. O., Mysterud, A., Gaillard, J. M., Tucker, C. J., & Stenseth, N. C. (2005). "Using the satellite-derived NDVI to assess ecological responses to environmental change." Trends in Ecology & Evolution, 20(9), 503-510.

[54] Ford, N. (2014). "Functional Thinking: Paradigm Over Syntax." O'Reilly Media.

[55] Lipovaca, M. (2011). "Learn You a Haskell for Great Good!: A Beginner's Guide." No Starch Press.

[56] Kühne, S., Reißner, M., & Rittmann, M. (2019). "Overcoming obstacles when transitioning to functional programming in Java projects." Journal of Systems and Software, 150, 190-207.

[57] Karacali, B., & Krim, H. (2003). "Fast minimization of structural risk by nearest neighbor rule." IEEE Transactions on Neural Networks, 14(1), 127-137.

[58] Wadler, P. (1998). "Why no one uses functional languages." ACM SIGPLAN Notices, 33(8), 23-27.

[59] Blelloch, G. E. (1990). "Prefix sums and their applications." Carnegie Mellon University.

[60] Neteler, M., & Mitasova, H. (2008). "Open source GIS: a GRASS GIS approach." Springer Science & Business Media.

[61] Warmerdam, F. (2008). "The geospatial data abstraction library." In Open Source Approaches in Spatial Data Handling (pp. 87-104). Springer.

[62] van der Maaten, L., & Hinton, G. (2008). "Visualizing data using t-SNE." Journal of Machine Learning Research, 9(Nov), 2579-2605.

[63] Marlow, S., Maier, P., Trinder, P. W., & Loidl, H. W. (2016). "Seq no more: Better strategies for parallel Haskell." Journal of Functional Programming, 26, e18.

[64] Peyton Jones, S. L., & Santos, A. L. (1998). "A transformation-based optimiser for Haskell." Science of Computer Programming, 32(1-3), 3-47.

[65] Hudak, P., Hughes, J., Peyton Jones, S., & Wadler, P. (2007). "A history of Haskell: being lazy with class." Proceedings of the third ACM SIGPLAN conference on History of programming languages, 12-1.

[66] Odersky, M., & Zenger, M. (2005). "Scalable component abstractions." ACM SIGPLAN Notices, 40(10), 41-57.

[67] Evans, E. (2004). "Domain-driven design: tackling complexity in the heart of software." Addison-Wesley Professional.

[68] Mernik, M., Heering, J., & Sloane, A. M. (2005). "When and how to develop domain-specific languages." ACM Computing Surveys, 37(4), 316-344.

[69] Hudak, P. (1996). "Building domain-specific embedded languages." ACM Computing Surveys, 28(4es), 196.

[70] Fowler, M. (2010). "Domain-specific languages." Pearson Education.

[71] Czaplicki, E. (2012). "Elm: Concurrent FRP for Functional GUIs." Senior thesis, Harvard University.

[72] Elliott, C., & Hudak, P. (1997). "Functional reactive animation." ACM SIGPLAN Notices, 32(8), 263-273.

[73] Wan, Z., & Hudak, P. (2000). "Functional reactive programming from first principles." ACM SIGPLAN Notices, 35(5), 242-252.

[74] Blackhurst, J., Craighead, C. W., Elkins, D., & Handfield, R. B. (2005). "An empirically derived agenda of critical research issues for managing supply-chain disruptions." International Journal of Production Research, 43(19), 4067-4081.

[75] Muller, C. L., Chapman, L., Johnston, S., Kidd, C., Illingworth, S., Foody, G., ... & Leigh, R. R. (2015). "Crowdsourcing for climate and atmospheric sciences: current status and future potential." International Journal of Climatology, 35(11), 3185-3203.

[76] Goodchild, M. F., & Glennon, J. A. (2010). "Crowdsourcing geographic information for disaster response: a research frontier." International Journal of Digital Earth, 3(3), 231-241.

[77] Tomkins, A., & Zhang, L. (2018). "Learning to rank and cluster at scale." ACM Transactions on Information Systems, 36(3), 1-34.

[78] Backus, J. (1978). "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs." Communications of the ACM, 21(8), 613-641.

[79] Wadler, P. (1995). "Monads for functional programming." Advanced Functional Programming, 24-52.

[80] Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., & Weizenbaum, N. (2010). "FlumeJava: easy, efficient data-parallel pipelines." ACM SIGPLAN Notices, 45(6), 363-375.

[81] Hinsen, K. (2009). "The promises of functional programming." Computing in Science & Engineering, 11(4), 86-90.