Course Outlines

1st Semester

DATABASES

Aims

This course introduces basic concepts for database systems as seen from the perspective of application designers. That is, the focus is on the abstractions supported by database management systems and not on how those abstractions are implemented.

The database world is currently undergoing swift and dramatic transformations largely driven by Internet-oriented applications and services. Today many more options are available to database application developers than in the past and so it is becoming increasingly difficult to sort fact from fiction. The course attempts to cut through the fog with a practical approach that emphasises engineering tradeoffs that underpin these recent developments and also guide our selection of "the right tool for the job."

This course covers three approaches. First, the traditional mainstay of the database industry -- the relational approach -- is described with emphasis on eliminating logical redundancy in data. Then two representatives of recent trends are presented -- graph-oriented and document-oriented databases. The lectures are supported with two Help and Tick sessions, where students gain hands-on experience and guidance with the Assessed Exercises (ticks).

Lectures

L1 Introduction. What is a database system? What is a data model? Typical DBMS configurations and variations (fast queries or reliable update). In-core, in secondary store or distributed.
L2 Conceptual modelling. Entities, relations, E/R diagrams and implementation-independent modelling, weak entities, cardinality.
L3+4 The relational database model. Implementing E/R models with relational tables. Relational algebra and SQL. Basic query primitives. Update anomalies caused by redundancy. Minimising redundancy with normalised schemas.
L5 Transactions. On-Line Transaction Processing. On-line Analytical Processing. Reliability, throughput, normal forms, ACID, BASE, eventual consistency.
L6 Documents and semi-structured data. The NoSQL, schema-free movement. XML/JSON. Key/value stores. Embracing data redundancy: representing data for fast, application-specific access. Path queries (if time permits).
L7 Further SQL. Multisets, NULL values, aggregates, transitive closure, expressibility (nested queries and recursive SQL).
L8 Graph databases. Optimised for processing enormous numbers of nodes and edges. Implementing E/R models in a graph-oriented database. Comparison of the presented models.

Objectives

At the end of the course students should

be able to design entity-relationship diagrams to represent simple database application scenarios
know how to convert entity-relationship diagrams to relational- and graph-oriented implementations
understand the fundamental tradeoff between the ease of updating data and the response time of complex queries
understand that no single data architecture can be used to meet all data management requirements
be familiar with recent trends in the database area.
Recommended reading
Lemahieu, W., Broucke, S. van den and Baesens, B. (2018) Principles of database management. Cambridge University Press.

DIGITAL ELECTRONICS

Aims
The aims of this course are to present the principles of combinational and sequential digital logic design and optimisation at a gate level. The use of n and p channel MOSFETs for building logic gates is also introduced.

Topics
Introduction. Semiconductors to computers. Logic variables. Examples of simple logic. Logic gates. Boolean algebra. De Morgan's theorem.
Logic minimisation. Truth tables and normal forms. Karnaugh maps. Quine-McCluskey method.
Binary adders. Half adder, full adder, ripple carry adder, fast carry generation.
Combinational logic design: further considerations. Multilevel logic. Gate propagation delay. An introduction to timing diagrams. Hazards and hazard elimination. Other ways to implement combinational logic.
Introduction to practical classes. Prototyping box. Breadboard and Dual in line (DIL) packages. Wiring.
Sequential logic. Memory elements. RS latch. Transparent D latch. Master-slave D flip-flop. T and JK flip-flops. Setup and hold times.
Sequential logic. Counters: Ripple and synchronous. Shift registers. System timing - setup time constraint, clock skew, metastability.
Synchronous State Machines. Moore and Mealy finite state machines (FSMs). Reset and self starting. State transition diagrams. Elimination of redundant states - row matching and state equivalence/implication table.
Further state machines. State assignment: sequential, sliding, shift register, one hot. Implementation of FSMs.
Introduction to microprocessors. Microarchitecture, fetch, register access, memory access, branching, execution time.
Electronics, Devices and Circuits. Current and voltage, conductors/insulators/semiconductors, resistance, basic circuit theory, the potential divider. Solving non-linear circuits. P-N junction (forward and reverse bias), N and p channel MOSFETs (operation and characteristics) and n-MOSFET logic, e.g., n-MOSFET inverter. Power consumption and switching time problems problems in n-MOSFET logic. CMOS logic (NOT, NAND and NOR gates), logic families, noise margin.

Objectives
At the end of the course students should

understand the relationships between combination logic and boolean algebra, and between sequential logic and finite state machines;
be able to design and minimise combinational logic;
appreciate tradeoffs in complexity and speed of combinational designs;
understand how state can be stored in a digital logic circuit;
know how to design a simple finite state machine from a specification and be able to implement this in gates and edge triggered flip-flops;

understand how to use MOSFETs to build digital logic circuits.
Recommended reading
* Harris, D.M. and Harris, S.L. (2013). Digital design and computer architecture. Morgan Kaufmann (2nd ed.). The first edition is still relevant.
Katz, R.H. (2004). Contemporary logic design. Benjamin/Cummings. The 1994 edition is more than sufficient.
Hayes, J.P. (1993). Introduction to digital logic design. Addison-Wesley.

Books for reference:

Horowitz, P. and Hill, W. (1989). The art of electronics. Cambridge University Press (2nd ed.) (more analog).
Weste, N.H.E. and Harris, D. (2005). CMOS VLSI Design - a circuits and systems perspective. Addison-Wesley (3rd ed.).
Mead, C. and Conway, L. (1980). Introduction to VLSI systems. Addison-Wesley.
Crowe, J. and Hayes-Gill, B. (1998). Introduction to digital electronics. Butterworth-Heinemann.
Gibson, J.R. (1992). Electronic logic circuits. Butterworth-Heinemann.

DISCRETE MATHEMATICS

Aims
The course aims to introduce the mathematics of discrete structures, showing it as an essential tool for computer science that can be clever and beautiful.

Lectures
Proof [5 lectures].
Proofs in practice and mathematical jargon. Mathematical statements: implication, bi-implication, universal quantification, conjunction, existential quantification, disjunction, negation. Logical deduction: proof strategies and patterns, scratch work, logical equivalences. Proof by contradiction. Divisibility and congruences. Fermat's Little Theorem.

Numbers [5 lectures].
Number systems: natural numbers, integers, rationals, modular integers. The Division Theorem and Algorithm. Modular arithmetic. Sets: membership and comprehension. The greatest common divisor, and Euclid's Algorithm and Theorem. The Extended Euclid's Algorithm and multiplicative inverses in modular arithmetic. The Diffie-Hellman cryptographic method. Mathematical induction: Binomial Theorem, Pascal's Triangle, Fundamental Theorem of Arithmetic, Euclid's infinity of primes.

Sets [9 lectures].
Extensionality Axiom: subsets and supersets. Separation Principle: Russell's Paradox, the empty set. Powerset Axiom: the powerset Boolean algebra, Venn and Hasse diagrams. Pairing Axiom: singletons, ordered pairs, products. Union axiom: big unions, big intersections, disjoint unions. Relations: composition, matrices, directed graphs, preorders and partial orders. Partial and (total) functions. Bijections: sections and retractions. Equivalence relations and set partitions. Calculus of bijections: characteristic (or indicator) functions. Finite cardinality and counting. Infinity axiom. Surjections. Enumerable and countable sets. Axiom of choice. Injections. Images: direct and inverse images. Replacement Axiom: set-indexed constructions. Set cardinality: Cantor-Schoeder-Bernstein Theorem, unbounded cardinality, diagonalisation, fixed-points. Foundation Axiom.

Formal languages and automata [5 lectures].
Introduction to inductive definitions using rules and proof by rule induction. Abstract syntax trees. Regular expressions and their algebra. Finite automata and regular languages: Kleene's theorem and the Pumping Lemma.

Objectives
On completing the course, students should be able to

prove and disprove mathematical statements using a variety of techniques;
apply the mathematical principle of induction;
know the basics of modular arithmetic and appreciate its role in cryptography;

understand and use the language of set theory in applications to computer science;
define sets inductively using rules and prove properties about them;
convert between regular expressions and finite automata;
use the Pumping Lemma to prove that a language is not regular.

Recommended reading

Biggs, N.L. (2002). Discrete mathematics. Oxford University Press (Second Edition).

Davenport, H. (2008). The higher arithmetic: an introduction to the theory of numbers. Cambridge University Press.

Hammack, R. (2013). Book of proof. Privately published (Second edition). Available at: http://www.people.vcu.edu/ rhammack/BookOfProof/index.html

Houston, K. (2009). How to think like a mathematician: a companion to undergraduate mathematics. Cambridge University Press.

Kozen, D.C. (1997). Automata and computability. Springer.

Lehman, E.; Leighton, F.T.; Meyer, A.R. (2014). Mathematics for computer science. Available on-line.

Velleman, D.J. (2006). How to prove it: a structured approach. Cambridge University Press (Second Edition).

FOUNDATIONS OF COMPUTER SCIENCE

Aims

The main aim of this course is to present the basic principles of programming. As the introductory course of the Computer Science Tripos, it caters for students from all backgrounds. To those who have had no programming experience, it will be comprehensible; to those experienced in languages such as C, it will attempt to correct any bad habits that they have learnt.

A further aim is to introduce the principles of data structures and algorithms. The course will emphasise the algorithmic side of programming, focusing on problem-solving rather than on hardware-level bits and bytes. Accordingly it will present basic algorithms for sorting, searching, etc., and discuss their efficiency using O-notation. Worked examples (such as polynomial arithmetic) will demonstrate how algorithmic ideas can be used to build efficient applications.

The course will use a functional language (OCaml). OCaml is particularly appropriate for inexperienced programmers, since a faulty program cannot crash and OCaml's unobtrusive type system captures many program faults before execution. The course will present the elements of functional programming, such as curried and higher-order functions. But it will also introduce traditional (procedural) programming, such as assignments, arrays and references.

Lectures

Introduction to Programming. The role of abstraction and representation. Introduction to integer and floating-point arithmetic. Declaring functions. Decisions and booleans. Example: integer exponentiation.
Recursion and Efficiency. Examples: Exponentiation and summing integers. Iteration versus recursion. Examples of growth rates. Dominance and O-Notation. The costs of some representative functions. Cost estimation.
Lists. Basic list operations. Append. Naïve versus efficient functions for length and reverse. Strings.
More on lists. The utilities take and drop. Pattern-matching: zip, unzip. A word on polymorphism. The "making change" example.
Sorting. A random number generator. Insertion sort, mergesort, quicksort. Their efficiency.
Datatypes and trees. Pattern-matching and case expressions. Exceptions. Binary tree traversal (conversion to lists): preorder, inorder, postorder.
Dictionaries and functional arrays. Functional arrays. Dictionaries: association lists (slow) versus binary search trees. Problems with unbalanced trees.
Functions as values. Nameless functions. Currying. The "apply to all" functional, map. Examples: The predicate functionals filter and exists.
Sequences, or lazy lists. Non-strict functions such as IF. Call-by-need versus call-by-name. Lazy lists. Their implementation in OCaml. Applications, for example Newton-Raphson square roots.
Queues and search strategies. Depth-first search and its limitations. Breadth-first search (BFS). Implementing BFS using lists. An efficient representation of queues. Importance of efficient data representation.

Elements of procedural programming. Address versus contents. Assignment versus binding. Own variables. Arrays, mutable or not. Introduction to linked lists.

Objectives

At the end of the course, students should

be able to write simple OCaml programs;

understand the fundamentals of using a data structure to represent some mathematical abstraction;

be able to estimate the efficiency of simple algorithms, using the notions of average-case, worse-case and amortised costs;

know the comparative advantages of insertion sort, quick sort and merge sort;

understand binary search and binary search trees;

know how to use currying and higher-order functions;

understand how OCaml combines imperative and functional programming in a single language.

Recommended reading

John Whitington. OCaml from the Very Beginning. (http://ocaml-book.com).

Okasaki, C. (1998). Purely functional data structures. Cambridge University Press.

HARDWARE PRACTICAL CLASSES

The Hardware Practical Classes accompany the Digital Electronics series of lectures. The aim of the Practical Classes is to enable students to get hands-on experience of designing, building, and testing and debugging of digital electronic circuits. The labs will take place in the Intel Lab. located in the William Gates Building (WGB).

The Digital Electronics lecture series occupies 12 lectures in the first 4 weeks of the Michaelmas Term, while the Practical Classes occupy the latter 6 weeks of the Michaelmas Term and the first 6 weeks of the Lent Term. If required, extra sessions will be available for any students needing to 'catch-up' on missed sessions or to complete any remaining practical work.

The Practical Classes take the form of 4 workshops, specifically:

Workshop 1 – Electronic Die;
Workshop 2 – Shaft Position Encoder;
Workshop 3 – Debouncing a Switch;
Workshop 4 – Framestore for an LED Array.
In general, the workshops require some preparatory work to be done prior to the practical session. These tasks are highlighted at the beginning of each Worksheet. Typically this involves preparing a design that you will then build, test and modify during the practical class. Note that insufficient preparation prior to the practical classes may compromise effective use of your time in the Intel lab.

In the Practical Classes you will usually work on your own, and you are expected to complete one Workshop in each of your scheduled sessions. Demonstrators are available during the sessions to assist you with any queries or problems you may have.

Important: Remember to get your work ticked.

INTRODUCTION TO GRAPHICS

Aims

To introduce the necessary background, the basic algorithms, and the applications of computer graphics and graphics hardware.

Lectures

Background. What is an image? Resolution and quantisation. Storage of images in memory. [1 lecture]

Rendering. Perspective. Reflection of light from surfaces and shading. Geometric models. Ray tracing. [2 lectures]

Graphics pipeline. Polygonal mesh models. Transformations using matrices in 2D and 3D. Homogeneous coordinates. Projection: orthographic and perspective. [1 lecture]

Graphics hardware and modern OpenGL. GPU rendering. GPU frameworks and APIs. Vertex processing. Rasterisation. Fragment processing. Working with meshes and textures. Z-buffer. Double-buffering and frame synchronization. [2 lectures]

Human vision, colour and tone mapping. Perception of colour. Tone mapping. Colour spaces. [2 lectures]

Objectives

By the end of the course students should be able to:

understand and apply in practice basic concepts of ray-tracing: ray-object intersection, reflections, refraction, shadow rays, distributed ray-tracing, direct and indirect illumination;

describe and explain the following algorithms: z-buffer, texture mapping, double buffering, mip-map, and normal-mapping;

use matrices and homogeneous coordinates to represent and perform 2D and 3D transformations; understand and use 3D to 2D projection, the viewing volume, and 3D clipping;

implement OpenGL code for rendering of polygonal objects, control camera and lighting, work with vertex and fragment shaders;

describe a number of colour spaces and their relative merits.

explain the need for tone mapping and colour processing in rendering pipeline.

Recommended reading

* Shirley, P. and Marschner, S. (2009). Fundamentals of Computer Graphics. CRC Press (3rd ed.).

Foley, J.D., van Dam, A., Feiner, S.K. and Hughes, J.F. (1990). Computer graphics: principles and practice. Addison-Wesley (2nd ed.).

Kessenich, J.M., Sellers, G. and Shreiner, D (2016). OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V, [seventh edition and later]

OBJECT-ORIENTED PROGRAMMING

Aims
The goal of this course is to provide students with an understanding of Object-Oriented Programming. Concepts are demonstrated in multiple languages, but the primary language is Java.

Lecture syllabus
Types, Objects and Classes Moving from functional to imperative. Functions, methods. Control flow. values, variables and types. Primitive Types. Classes as custom types. Objects vs Classes. Class definition, constructors. Static data and methods.
Designing Classes Identifying classes. UML class diagrams. Modularity. Encapsulation/data hiding. Immutability. Access modifiers. Parameterised types (Generics).
Pointers, References and Memory Pointers and references. Reference types in Java. The call stack. The heap. Iteration and recursion. Pass-by-value and pass-by-reference.
Inheritance Inheritance. Casting. Shadowing. Overloading. Overriding. Abstract Methods and Classes.
Polymorphism and Multiple Inheritance Polymorphism in ML and Java. Multiple inheritance. Interfaces in Java.
Lifecycle of an Object Constructors and chaining. Destructors. Finalizers. Garbage Collection: reference counting, tracing.
Java Collections and Object Comparison Java Collection interface. Key classes. Collections class. Iteration options and the use of Iterator. Comparing primitives and objects. Operator overloading.
Error Handling Types of errors. Limitations of return values. Deferred error handling. Exceptions. Custom exceptions. Checked vs unchecked. Inappropriate use of exceptions. Assertions.
DesignLanguage evolution Need for languages to evolve. Generics in Java. Type erasure. Introduction to Java 8: Lambda functions, functions as values, method references, streams.
Design Patterns Introduction to design patterns. Open-closed principle. Examples of Singleton, Decorator, State, Composite, Strategy, Observer. [2 lectures]
Objectives
At the end of the course students should

Provide an overview of key OOP concepts that are transferable in mainstream programming languages;
Gain an understanding of software development approaches adopted in the industry including maintainability, testing, and software design patterns;
Increase programming familiarity with Java;
Recommended reading
1. Real-World Software Development: A Project-Driven Guide to Fundamentals in Java [Urma et al.]
2. Modern Java in Action (2nd edition) [Urma et al.].
3. Java How to Program: early objects [Deitel et al.]

OCAML PRACTICAL CLASSES

The Role of Tickers
A ticking session is a short (5 minute) one-to-one session with a ticker conducted on Thursday afternoons. The role of the ticker is twofold:

to check you understand your solution (and didn't just have some lucky guesses or copy code from elsewhere)
to give you general feedback on ways to improve your code.
To those ends a Ticker will typically ask you questions related to the core material of the tick and discuss your code directly.

Deadline Extensions
Deadline extensions can be granted for illness or similar reasons. To obtain an extension please email Jon Ludlam in the first instance, clearly stating your justification for an extension and CCing your Director of Studies, who will need to support your request.

SCIENTIFIC COMPUTING

Aims

This course is a hands-on introduction to using computers to investigate scientific models and data.

Syllabus

Python notebooks. Overview of the Python programming language. Use of notebooks for scientific computing.

Numerical computation. Writing fast vectorized code in numpy. Optimization and fitting. Simulation.

Working with data. Data import. Common ways to summarize and plot data, for univariate and multivariate analysis.

Objectives

At the end of the course students should

be able to import data, plot it, and summarize it appropriately

be able to write fast vectorized code for scientific / data work

2nd Semester

ALGORITHMS 1

Aims

The aim of this course is to provide an introduction to computer algorithms and data structures, with an emphasis on foundational material.

Lectures

Sorting. Review of complexity and O-notation. Trivial sorting algorithms of quadratic complexity. Review of merge sort and quicksort, understanding their memory behaviour on statically allocated arrays. Heapsort. Stability. Other sorting methods including sorting in linear time. Median and order statistics. [Ref: CLRS3 chapters 1, 2, 3, 6, 7, 8, 9] [about 4 lectures]
Strategies for algorithm design. Dynamic programming, divide and conquer, greedy algorithms and other useful paradigms. [Ref: CLRS3 chapters 4, 15, 16] [about 3 lectures]
Data structures. Elementary data structures: pointers, objects, stacks, queues, lists, trees. Binary search trees. Red-black trees. B-trees. Hash tables. Priority queues and heaps. [Ref: CLRS3 chapters 6, 10, 11, 12, 13, 18] [about 5 lectures].

Objectives

At the end of the course students should:

have a thorough understanding of several classical algorithms and data structures;
be able to analyse the space and time efficiency of most algorithms;
have a good understanding of how a smart choice of data structures may be used to increase the efficiency of particular algorithms;
be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result.

Recommended reading

* Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein Introduction to Algorithms, Fourth Edition ISBN 9780262046305 Published: April 5, 2022.

Sedgewick, R., Wayne, K. (2011). Algorithms. Addison-Wesley. ISBN 978-0-321-57351-3.

Kleinberg, J. and Tardos, É. (2006). Algorithm design. Addison-Wesley. ISBN 978-0-321-29535-4.

Knuth, D.A. (2011). The Art of Computer Programming. Addison-Wesley. ISBN 978-0-321-75104-1.

ALGORITHMS 2

Aims
The aim of this course is to provide an introduction to computer algorithms and data structures, with an emphasis on foundational material.

Lectures
Graphs and path-finding algorithms. Graph representations. Breadth-first and depth-first search. Single-source shortest paths: Bellman-Ford and Dijkstra algorithms. All-pairs shortest paths: dynamic programming and Johnson's algorithms. [About 4 lectures]
Graphs and subgraphs. Maximum flow: Ford-Fulkerson method, Max-flow min-cut theorem. Matchings in bipartite graphs. Minimum spanning tree: Kruskal and Prim algorithms. Topological sort. [About 4 lectures]
Advanced data structures. Binomial heap. Amortized analysis: aggregate analysis, potential method. Fibonacci heaps. Disjoint sets. [About 4 lectures]
Objectives
At the end of the course students should:

have a thorough understanding of several classical algorithms and data structures;
be able to analyse the space and time efficiency of most algorithms;
have a good understanding of how a smart choice of data structures may be used to increase the efficiency of particular algorithms;
be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result.
Recommended reading
* Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein Introduction to Algorithms, Fourth Edition ISBN 9780262046305 Published: April 5, 2022.

Sedgewick, R., Wayne, K. (2011). Algorithms. Addison-Wesley. ISBN 978-0-321-57351-3.

Kleinberg, J. and Tardos, É. (2006). Algorithm design. Addison-Wesley. ISBN 978-0-321-29535-4.

Knuth, D.A. (2011). The Art of Computer Programming. Addison-Wesley. ISBN 978-0-321-75104-1.

MACHINE LEARNING AND REAL-WORLD DATA

Aims
This course introduces students to machine learning algorithms as used in real-world applications, and to the experimental methodology necessary to perform statistical analysis of large-scale data from unpredictable processes. Students will perform 3 extended practicals, as follows:

Statistical classification: Determining movie review sentiment using Naive Bayes;
Sequence Analysis: Hidden Markov Modelling and its application to a task from biology (predicting protein interactions with a cell membrane);
Analysis of social networks, including detection of cliques and central nodes.
Syllabus
Topic One: Statistical Classification
Introduction to sentiment classification.
Naive Bayes parameter estimation.
Statistical laws of language.
Statistical tests for classification tasks.
Cross-validation and test sets.
Uncertainty and human agreement.
Topic Two: Sequence Analysis
Hidden Markov Models (HMM) and HMM training.
The Viterbi algorithm.
Using an HMM in a biological application.
Topic Three: Social Networks
Properties of networks: Degree, Diameter.
Betweenness Centrality.
Clustering using betweenness centrality.
Objectives
By the end of the course students should be able to:

understand and program two simple supervised machine learning algorithms;
use these algorithms in statistically valid experiments, including the design of baselines, evaluation metrics, statistical testing of results, and provision against overtraining;
visualise the connectivity and centrality in large networks;
use clustering (i.e., a type of unsupervised machine learning) for detection of cliques in unstructured networks.
Recommended reading
Jurafsky, D. and Martin, J. (2008). Speech and language processing. Prentice Hall.

Easley, D. and Kleinberg, J. (2010). Networks, crowds, and markets: reasoning about a highly connected world. Cambridge University Press.

OPERATING SYSTEMS

Aims
The overall aim of this course is to provide a general understanding of the structure and key functions of the operating system. Case studies will be used to illustrate and reinforce fundamental concepts.

Lectures
Introduction to operating systems. Elementary computer organisaton. Abstract view of an operating system. Key concepts: layering, multiplexing, performance, caching, buffering, policies versus mechanisms. OS evolution: multi-programming, time-sharing. Booting. [1 lecture]
Protection. Dual-mode operation. Protecting CPU, memory, I/O, storage. Kernels, micro-kernels, and modules. System calls. Virtual machines and containers. Subjects and objects. Authentication. Access matrix: ACLs and capabilities. Covert channels. [1 lecture]
Processes. Job/process concepts. Process lifecycle. Process management. Context switching. Inter-process communication. [1 lecture]
Scheduling. CPU-I/O burst cycle. Scheduling basics: preemption and non-preemption. Scheduling algorithms: FCFS, SJF, SRTF, Priority, and Round Robin. Multilevel scheduling. [2 lectures]
Memory management. Processes in memory. Logical versus physical addresses. Partitions: static versus dynamic, free space management, external fragmentation. Segmented memory. Paged memory: concepts, internal fragmentation, page tables. Demand paging/segmentation. Page replacement strategies: FIFO, OPT, and LRU with approximations including NRU, LFU, MFU, MRU. Working set schemes. Thrashing. [3 lectures]
I/O subsystem. General structure. Polled mode versus interrupt-driven I/O. Programmed I/O (PIO) versus Direct Memory Access (DMA). Application I/O interface: block and character devices, buffering, blocking, non-blocking, asynchronous, and vectored I/O. Other issues: caching, scheduling, spooling, performance. [1 lecture]
File management. File concept. Directory and storage services. File names and metadata. Directory name-space: hierarchies, DAGs, hard and soft links. File operations. Access control. Existence and concurrency control. [1 lecture]
Unix case study. History. General structure. Unix file system: file abstraction, directories, mount points, implementation details. Processes: memory image, lifecycle, start of day. The shell: basic operation, commands, standard I/O, redirection, pipes, signals. Character and block I/O. Process scheduling. [2 lectures]
Objectives
At the end of the course students should be able to:

describe the general structure and purpose of an operating system;
explain the concepts of process, address space, and file;
compare and contrast various CPU scheduling algorithms;
compare and contrast mechanisms involved in memory and storage management;
compare and contrast polled, interrupt-driven, and DMA-based access to I/O devices.
Recommended reading

* Silberschatz, A., Galvin, P.C., and Gagne, G. (2018). Operating systems concepts. Wiley (10th ed.). Prior editions, at least back to 8th ed., should also be acceptable.
Anderson, T. and Dahlin, M. (2014). Operating Systems: Principles and Practice. Recursive Books (2nd ed.).
Bacon, J. and Harris, T. (2003). Operating systems. Addison-Wesley (3rd ed.). Currently appears out-of-print.
Leffler, S. (1989). The design and implementation of the 4.3BSD Unix operating system. Addison-Wesley.
McKusick, M.K., Neville-Neil, G.N. and Watson, R.N.M. (2014) The Design and Implementation of the FreeBSD Operating System. Pearson Education. (2nd ed.).

INTERACTION DESIGN
Aims
The aim of this course is to provide an introduction to interaction design, with an emphasis on understanding and experiencing the user-centred design process, from conducting user research and requirements development to implementation and evaluation, while understanding the background to human-computer interaction.

Lectures
Course overview and user research methods. Introduction to the course and practicals.
User-Centred Design. User research methods.
Keeping usera in mind. User research data analysis. Identifying users and stakeholders.
Representing user goals and activities. Identifying and establishing requirements.
Design and prototyping. Methods for exploring the design space. Prototyping and different kinds of prototypes.
Visual and interaction design. Memory, perception, attention, and their implications for interaction design. Modalities of interaction, interaction design patterns, information architecture, and their implications for interaction design.
Evaluation. Practical methods for evaluating designs. Evaluation methods without users. Evaluation methods with users.
Case studies from industry and research. Guest lectures (the topics of these lectures are subject to change).
Objectives
By the end of the course students should

have a thorough understanding of the user-centred design process and be able to apply it to interaction design;
be able to design new user interfaces that are informed by principles of good design, and the principles of human visual perception, cognition and communication;
be able to prototype and implement interactive user interfaces with a strong emphasis on users, usability and appearance;
be able to evaluate existing or new user interfaces using multiple techniques;
be able to compare and contrast different design techniques and to critique their applicability to new domains.
Recommended reading
* Preece, J., Rogers, Y. and Sharp, H. (2015). Interaction design. Wiley (4th ed.).

INTRODUCTION TO PROBABILITY

Aims
This course provides an elementary introduction to probability and statistics with applications. Probability theory and the related field of statistical inference provide the foundations for analysing and making sense of data. The focus of this course is to introduce the language and core concepts of probability theory. The course will also cover some applications of statistical inference and algorithms in order to equip students with the ability to represent problems using these concepts and analyse the data within these principles.

Lectures
Part 1 - Introduction to Probability

Introduction. Counting/Combinatorics (revision), Probability Space, Axioms, Union Bound.
Conditional probability. Conditional Probabilities and Independence, Bayes'Theorem, Partition Theorem
Part 2 - Discrete Random Variables

Random variables. Definition of a Random Variable, Probability Mass Function, Cumulative Distribution, Expectation.
Probability distributions. Definition and Properties of Expectation, Variance, different ways of computing them, Examples of important Distributions (Bernoulli, Binomial, Geometric, Poisson), Primer on Continuous Distributions including Normal and Exponential Distributions.
Multivariate distributions. Multiple Random Variables, Joint and Marginal Distributions, Independence of Random Variables, Covariance.
Part 3 - Moments and Limit Theorems

Introduction. Law of Average, Useful inequalities (Markov and Chebyshef), Weak Law of Large Numbers (including Proof using Chebyshef's inequality), Examples.
Moments and Central Limit Theorem. Introduction to Moments of Random Variables, Central Limit Theorem (Proof using Moment Generating functions), Example.
Part 4 - Applications/Statistics

Statistics. Classical Parameter Estimation (Maximum-Likelihood-Estimation), bias, sample mean, sample variance), Examples (Collision-Sampling, Estimating Population Size).
Algorithms. Online Algorithms (Secretary Problem, Odd's Algorithm).
Objectives
At the end of the course students should

understand the basic principles of probability spaces and random variables
be able to formulate problems using concepts from probability theory and compute or estimate probabilities
be familiar with more advanced concepts such as moments, limit theorems and applications such as parameter estimation

Recommended reading

* Ross, S.M. (2014). A First course in probability. Pearson (9th ed.).

Bertsekas, D.P. and Tsitsiklis, J.N. (2008). Introduction to probability. Athena Scientific.

Grimmett, G. and Welsh, D. (2014). Probability: an Introduction. Oxford University Press (2nd ed.).

Dekking, F.M., et. al. (2005) A modern introduction to probability and statistics. Springer.

SOFTWARE AND SECURITY ENGINEERING

Aims

This course aims to introduce students to software and security engineering, and in particular to the problems of building large systems, safety-critical systems and systems that must withstand attack by capable opponents. Case histories of failure are used to illustrate what can go wrong, while current software and security engineering practice is studied as a guide to how failures can be avoided.

Lectures

1. What is a security policy or a safety case? Definitions and examples; one-way flows for both confidentiality and safety properties; separation of duties. Top-down and bottom-up analysis methods. What architecture can do, versus benefits of decoupling policy from mechanism.

2. Examples of safety and security policies. Safety and security usability; the pyramid of harms. Predicting and mitigating user errors. The prevention of fraud and error in accounting systems; the safety usability of medical devices.

3. Attitudes to risk: expected utility, prospect theory, framing, status quo bias. Authority, conformity and gender; mental models, affordances and defaults. The characteristics of human memory; forgetting passwords versus guessing them.

4. Security protocols; how to enforce policy using structured human interaction, cryptography or both. Middleperson attacks.The role of verification and its limitations.

5. Attacks on TLS, from rogue CAs through side channels to Heartbleed. Other types of software bugs: syntactic, timing, concurrency, code injection, buffer overflows. Defensive programming: secure coding, contracts. Fuzzing.

6. The software crisis. Examples of large-scale project failure, such as the London Ambulance Service system and the NHS National Programme for IT. Intrinsic difficulties with complex software.

7. Software engineering as the management of complexity. The software life cycle; requirements analysis methods; modular design; the role of prototyping; the waterfall, spiral and agile models.

8. The economics of software as a Service (SaaS); the impact SaaS has on software engineering. Continuous integration, release engineering, behavioural analytics and experiment frameworks, rearchitecting systems while in operation.

9. Critical systems: safety as an emergent system property. Examples of catastrophic failure: from Therac-25 to the Boeing 737Max. The problems of managing redundancy. The overall process of safety engineering.

10. Managing the development of critical systems: tools and methods, individual versus group productivity, economics of testing and agile development, measuring outcomes versus process, the technical and human aspects of management, post-market surveillance and coordinated disclosure. The sustainability of products with software components.

At the end of the course students should know how writing programs with tough assurance targets, in large teams, or both, differs from the programming exercises they have engaged in so far. They should understand the different models of software development described in the course as well as the value of various development and management tools. They should understand the development life cycle and its basic economics. They should understand the various types of bugs, vulnerabilities and hazards, how to find them, and how to avoid introducing them. Finally, they should be prepared for the organizational aspects of their Part IB group project.

Recommended reading
Anderson, R. (Third Edition 2020). Security engineering (Part 1 and Chapters 27-28). Wiley. Available at: http://www.cl.cam.ac.uk/users/rja14/book.html

3rd Semester
CONCURRENT AND DISTRIBUTED SYSTEMS

Aims
This course considers two closely related topics, Concurrent Systems and Distributed Systems, over 16 lectures. The aim of the first half of the course is to introduce concurrency control concepts and their implications for system design and implementation. The aims of the latter half of the course are to study the fundamental characteristics of distributed systems, including their models and architectures; the implications for software design; some of the techniques that have been used to build them; and the resulting details of good distributed algorithms and applications.

Lectures: Concurrency
Introduction to concurrency, threads, and mutual exclusion. Introduction to concurrent systems; threads; interleaving; preemption; parallelism; execution orderings; processes and threads; kernel vs. user threads; M:N threads; atomicity; mutual exclusion; and mutual exclusion locks (mutexes).
Automata Composition. Synchronous and asynchronous parallelism; sequential consistency; rendezvous. Safety, liveness and deadlock; the Dining Philosophers; Hardware foundations for atomicity: test-and-set, load-linked/store-conditional and fence instructions. Lamport bakery algorithm.
Common design patterns: semaphores, producer-consumer, and MRSW. Locks and invariants; semaphores; condition synchronisation; N-resource allocation; two-party and generalised producer-consumer; Multi-Reader, Single-Write (MRSW) locks.
CCR, monitors, and concurrency in practice. Conditional critical regions (CCR); monitors; condition variables; signal-wait vs. signal-continue semantics; concurrency in practice (kernels, pthreads, Java, Cilk, OpenMP).
Deadlock and liveness guarantees Offline vs. online; model checking; resource allocation graphs; lock order checking; deadlock prevention, avoidance, detection, and recovery; livelock; priority inversion; auto parallelisation.
Concurrency without shared data; transactions. Active objects; message passing; tuple spaces; CSP; and actor models. Composite operations; transactions; ACID; isolation; and serialisability.
Further transactions History graphs; good and bad schedules; isolation vs. strict isolation; 2-phase locking; rollback; timestamp ordering (TSO); and optimistic concurrency control (OCC).
Crash recovery, lock-free programming, and transactional memory. Write-ahead logging, checkpoints, and recovery. Lock-free programming. Hardware and software transactional memories.

Lectures: Distributed Systems
Introduction to distributed systems; RPC. Avantages and challenges of distributed systems; unbounded delay and partial failure; network protocols; transparency; client-server systems; remote procedure call (RPC); marshalling; interface definition languages (IDLs).

System models and faults. Synchronous, partially synchronous, and asynchronous network models; crash-stop, crash-recovery, and Byzantine faults; failures, faults, and fault tolerance; two generals problem.
Time, clocks, and ordering of events. Physical clocks; leap seconds; UTC; clock synchronisation and drift; Network Time Protocol (NTP). Causality; happens-before relation.
Logical time; Lamport clocks; vector clocks. Broadcast (FIFO, causal, total order); gossip protocols.
Replication. Quorums; idempotence; replica consistency; read-after-write consistency. State machine replication; leader-based replication.
Consensus and total order broadcast. FLP result; leader election; the Raft consensus algorithm.
Replica consistency. Two-phase commit; relationship between 2PC and consensus; linearizability; ABD algorithm; eventual consistency; CAP theorem.
Case studies. Conflict-free Replicated Data Types (CRDTs); collaborative text editing. Google's Spanner; TrueTime.
Objectives
At the end of Concurrent Systems portion of the course, students should:

understand the need for concurrency control in operating systems and applications, both mutual exclusion and condition synchronisation;
understand how multi-threading can be supported and the implications of different approaches;
be familiar with the support offered by various programming languages for concurrency control and be able to judge the scope, performance implications and possible applications of the various approaches;
be aware that dynamic resource allocation can lead to deadlock;
understand the concept of transaction; the properties of transactions, how they can be implemented, and how their performance can be optimised based on optimistic assumptions;
understand how the persistence properties of transactions are addressed through logging; and
have a high-level understanding of the evolution of software use of concurrency in the operating-system kernel case study.
At the end of the Distributed Systems portion of the course, students should:

understand the difference between shared-memory concurrency and distributed systems;
understand the fundamental properties of distributed systems and their implications for system design;
understand notions of time, including logical clocks, vector clocks, and physical time synchronisation;
be familiar with various approaches to data and service replication, as well as the concept of replica consistency;
understand the effects of large scale on the provision of fundamental services and the tradeoffs arising from scale;
appreciate the implications of individual node and network communications failures on distributed computation;
be aware of a variety of programming models and abstractions for distributed systems, such as RPC, middleware, and total order broadcast;

be familiar with a range of distributed algorithms, such as consensus, causal broadcast, and two-phase commit.
Recommended reading
Modern Operating Systems (free PDF available online) by Andrew S Tanenbaum, Herbert Bos
Java Concurrency in Practice' (2006 but still highly relevant) by Brian Goetz
Kleppmann, M. (2017). Designing data-intensive applications. O'Reilly.
Tanenbaum, A.S. and van Steen, M. (2017). Distributed systems, 3rd edition. available online.
Cachin, C., Guerraoui, R. and Rodrigues, L. (2011) Introduction to Reliable and Secure Distributed Programming. Springer (2nd edition).

DATA SCIENCE

Aims
This course introduces fundamental tools for describing and reasoning about data. There are two themes: designing probability models to describe systems; and drawing conclusions based on data generated by such systems.

Lectures
Specifying and fitting probability models. Random variables. Maximum likelihood estimation. Generative and supervised models. Goodness of fit.
Feature spaces. Vector spaces, bases, inner products, projection. Linear models. Model fitting as projection. Design of features.
Handling probability models. Handling pdf and cdf. Bayes’s rule. Monte Carlo estimation. Empirical distribution.
Inference. Bayesianism. Frequentist confidence intervals, hypothesis testing. Bootstrap resampling.
Random processes. Markov chains. Stationarity, and drift analysis. Processes with memory. Learning a random process.
Objectives
At the end of the course students should

be able to formulate basic probabilistic models, including discrete time Markov chains and linear models
be familiar with common random variables and their uses, and with the use of empirical distributions rather than formulae
understand different types of inference about noisy data, including model fitting, hypothesis testing, and making predictions
understand the fundamental properties of inner product spaces and orthonormal systems, and their application to modelling
Recommended reading
* F.M. Dekking, C. Kraaikamp, H.P. Lopuhaä, L.E. Meester (2005). A modern introduction to probability and statistics: understanding why and how. Springer.

S.M. Ross (2002). Probability models for computer science. Harcourt / Academic Press.

M. Mitzenmacher and E. Upfal (2005). Probability and computing: randomized algorithms and probabilistic analysis. Cambridge University Press.

ECAD AND ARCHITECTURE PRACTICAL CLASSES
Aims
The aims of this course are to enable students to apply the concepts learned in the Computer Design course. In particular a web based tutor is used to introduce the SystemVerilog hardware description language, while the remaining practical classes will then allow students to implement the design of components in this language.

Practical Classes
Web tutor The first class uses a web based tutor to rapidly teach the SystemVerilog language.
FPGA design flow Test driven hardware development for FPGA including an embedded processor and peripherals [3 classes]
Embedded system implementation Embedded system implementation on FPGA [3-4 classes]
Objectives
Gain experience in electronic computer aided design (ECAD) through learning a design-flow for field programmable gate arrays (FPGAs).
Learn how to interface to peripherals like a touch screen.
Learn how to debug hardware and software systems in simulation.
Understand how to construct and program a heterogeneous embedded system.
Recommended reading
* Harris, D.M. and Harris, S.L. (2007). Digital design and computer architecture: from gates to processors. Morgan Kaufmann.

Pointers to sources of more specialist information are included on the associated course web page.

Introduction
The ECAD and Architecture Laboratory sessions are a companion to the Introduction to Computer Architecture course. The objective is to provide experience of hardware design for FPGA including use of a small embedded processor. It covers hardware design in SystemVerilog, embedded software design in RISC-V assembly and C, and use of FPGA tools.

Prerequisites
This course assumes familiarity with the material from Part IA Digital Electronics, although we will use automated tools to perform many of the steps you might have done there by hand (for example, logic minimisation).

We will be using a Linux command-line environment. We provide scripts and guidance on what you need, however you may find it useful to review Unix Tools, in particular basic navigation such as ls, cd, mkdir, cp, mv. We'll be using the GCC compiler and Makefiles, described in parts 31-32. We will also be using git for revision control and submission of work for ticking - you should review at least parts 23-25 and 29.

Practicalities

The course runs over the 8 weeks of Michaelmas term. The course has been designed so you can do most of the work at home or at any time you wish. You should be able to run all the necessary tools on your own machine, through the use of virtual machines and Docker containers. We have a number of routes to do this in case some of them aren't suitable for the machine you have.

We're running the course in a hybrid mode this year. We will provide online help sessions via Microsoft Teams as well as in-person lab sessions on Fridays and Tuesdays 2-5pm during term.

The core components of the course, being the RISC-V architecture and software and ticked exercise, can be done entirely online, without needing access to hardware. They can be undertaken with any of the platforms we support, including MCS Linux (on the Intel Lab workstations and remotely) if your own machine isn't suitable. We expect everyone to complete these parts.

The rest of the course is optional. The hardware simulation and FPGA compilation can be done on your machine without access to hardware, if you are able to run the virtual machine.

The parts that require access to an FPGA board will need the ability to run the virtual machine and you be able to collect and return an FPGA board from the Department. That process will be described when you come to those parts. There is an optional starred tick available for completion of this part, which will need assessment by a demonstrator in person.

We'll do our best to support all the different platforms and variations, but please bear with us - it's quite possible we'll come across a problem we haven't seen before!

We have provided four routes you can use different tools to complete the course. Some routes necessarily exclude some material but this material is optional.

Assessment
In a similar way to other practical courses, this course carries one 'tick'. Everyone should expect to receive this tick, and those who do not should expect to lose marks on their final exams.

This year we have adopted the 'chime' autoticker used by Further Java. You will check out the initial files as a git repository from the chime server, commit your work, and push the repository back to submit it. You can then press a button to run tests against your code and the autoticker will tell you whether you have passed. Additionally you may be selected for a (real/video) interview with a ticker to discuss your code and understanding of the material.

Ticking procedures will only be available during weeks 1-8 of Michaelmas term. The deadline for submitting Tick 1 is 5pm on the Tuesday of week 6 (19 November 2024). After passing the autoticker you may be asked for a tick interview with a demonstrator (online or in person). You can gain Tick 1* during the timetabled lab sessions, assuming demonstrators are available.

If you do not submit a successful Tick 1 to the autoticker by 5pm on the Tuesday of week 6 you can self-certify an extension. If you need more time than this you will need to ask your Directory of Studies to organise a further extension. The hard deadline by which all ticks must be completed is noon on Friday 31 January 2025 (the Head of Department Notices is the definitive document). Extensions beyond this date due to exceptional circumstances require a formal application from your college to the Examiners.

Scheduling
While the full course contains 8 exercises, there is not a tight binding to doing one exercise per week. Students should expect to spend about three hours per week on the course, however it is recommended that you make a start on the next exercise if you have time in hand.

Demonstrator help will be focused on the Tuesday and Friday 2-5pm slots during term the lab would normally operate in, so you may find it helpful to work in these times as that will provide the quickest response to questions.

Collaboration
When we have done these labs in person, we have often found they worked well when doing them collaboratively. While your work must be your own, students can work together and help each other, and demonstrators contribute advice and experience with the tools.

For the timetabled sessions, which are Tuesdays and Fridays 2-5pm UK time during term, there will be demonstrators available in the lab and at the same time we'll use the ECAD group on Microsoft Teams. With this we can provide online audio/video help, screensharing and (for the Windows and possibly Mac Teams apps) optional remote control of your development environment. You will be added to the Team in week 1 - if you believe you have been missed please post in the Moodle forum. In Teams there are a number of Helpdesk Channels (A-C) - if you need help during lab times, post in Help Centre and a demonstrator can ask you to go to a specific channel. There are also Student Breakout Channels (D-F) which are available for students to chat amongst themselves.

For help outside of timetabled sessions we've set up a Moodle forum where you can post questions - we'll keep an eye on this at other times, but students are encouraged to use it to support each other.

Students are of course free to use other platforms to help each other. If you find anything useful that we might use in future, please let us know!

If you are having difficulty accessing both Moodle and Teams, please email theo.markettos at cl.cam.ac.uk.

Feedback
We revise the course each year, which may cause new bugs in the code or the notes. The creators (Theo Markettos and Simon Moore) appreciate constructive feedback.

Credits

This course was written by Theo Markettos and Simon Moore, with portions developed by Robert Eady, Jonas Fiala, Paul Fox, Vlad Gavrila, Brian Jones and Roy Spliet. We would like to thank Altera, Intel and Terasic for funding and other contributions.

ECONOMICS, LAW AND ETHICS

Aims
This course aims to give students an introduction to some basic concepts in economics, law and ethics.

Lectures
Classical economics and consumer theory. Prices and markets; Pareto efficiency; preferences; utility; supply and demand; the marginalist revolution; elasticity; the welfare theorems; transaction costs.
Information economics. The discriminating monopolist; marginal costs; effects of technology on supply and demand; competition and information; lock in; real and virtual networks; Metcalfe's law; the dominant firm model; price discrimination; bundling; income distribution.
Market failure and behavioural economics. Market failure: the business cycle; recession and technology; tragedy of the commons; externalities; monopoly rents; asymmetric information: the market for lemons; adverse selection; moral hazard; signalling. Behavioural economics: bounded rationality, heuristics and biases; nudge theory; the power of defaults; agency effects.
Auction theory and game theory. Auction theory: types of auctions; strategic equivalence; the revenue equivalence theorem; the winner's curse; problems with real auctions; mechanism design and the combinatorial auction; applicability of auction mechanisms in computer science; advertising auctions. Game theory: the choice between cooperation and conflict; strategic forms; dominant strategy equilibrium; Nash equilibrium; the prisoners' dilemma; evolution of strategies; stag hunt; volunteer's dilemma; chicken; iterated games; hawk-dove; application to computer science.
Principles of law. Criminal and civil law; contract law; choice of law and jurisdiction; arbitration; tort; negligence; defamation; intellectual property rights.
Law and the Internet. Computer evidence; the General Data Protection Regulation; UK laws that specifically affect the Internet; e-commerce regulations; privacy and electronic communications.
Philosophies of ethics. Authority, intuitionist, egoist and deontological theories; utilitarian and Rawlsian models; morality; insights from evolutionary psychology, neurology, and experimental ethics; professional codes of ethics; research ethics.
Contemporary ethical issues. The Internet and social policy; current debates on privacy, surveillance, and censorship; responsible vulnerability disclosure; algorithmic bias; predictive policing; gamification and engagement; targeted political advertising; environmental impacts.
Objectives
On completion of this course, students should be able to:

Reflect on and discuss professional, economic, social, environmental, moral and ethical issues relating to computer science
Define and explain economic and legal terminology and arguments
Apply the philosophies and theories covered to computer science problems and scenarios
Reflect on the main constraints that market, legislation and ethics place on firms dealing in information goods and services
Recommended reading

* Shapiro, C. & Varian, H. (1998). Information rules. Harvard Business School Press.
Hare, S. (2022). Technology is not neutral: A short guide to technology ethics. London Publishing Partnership.

Further reading:
Smith, A. (1776). An inquiry into the nature and causes of the wealth of nations, available at http://www.econlib.org/library/Smith/smWN.html
Thaler, R.H. (2016). Misbehaving. Penguin.
Galbraith, J.K. (1991). A history of economics. Penguin.
Poundstone, W. (1992). Prisoner’s dilemma. Anchor Books.
Pinker, S (2011). The Better Angels of our Nature. Penguin.
Anderson, R. (2008). Security engineering (Chapter 7). Wiley.
Varian, H. (1999). Intermediate microeconomics - a modern approach. Norton.
Nuffield Council on Bioethics (2015) The collection, linking and use of data in biomedical research and health care.

FURTHER GRAPHICS

Aims
The course introduces fundamental concepts of modern graphics pipelines.

Lectures
The order and content of lectures is provisional and subject to change.

Geometry Representations. Parametric surfaces, implicit surfaces, meshes, point-set surfaces, geometry processing pipeline. [1 lecture]
Discrete Differential Geometry. Surface normal and curvature, Laplace-Beltrami operator, heat diffusion. [1 lecture]
Geometry Processing. Parametrization, filtering, 3D capture. [1 lecture]
Animation I. Animation types, animation pipeline, rigging/skinning, character animation. [1 lecture]
Animation II. Blending transformations, pose-space animation, controllers. [1 lecture]
The Rendering Equation. Radiosity, reflection models, BRDFs, local vs. global illumination. [1 lecture]
Distributed Ray Tracing. Quadrature, importance sampling, recursive ray tracing. [1 lecture]
Inverse Rendering. Differentiable rendering, inverse problems. [1 lecture]
Objectives
On completing the course, students should be able to

understand and use fundamental 3D geometry/scene representations and operations;
learn animation techniques and controls;
learn how light transport is modeled and simulated in rendering;
understand how graphics and rendering can be used to solve computer perception problems.
Recommended reading
Students should expect to refer to one or more of these books, but should not find it necessary to purchase any of them.

Shirley, P. and Marschner, S. (2009). Fundamentals of Computer Graphics. CRC Press (3rd ed.).
Watt, A. (2000). 3D Computer Graphics. Addison-Wesley (3rd ed).
Hughes, van Dam, McGuire, Skalar, Foley, Feiner and Akeley (2013). Computer Graphics: Principles and Practice. Addison-Wesley (3rd edition)
Akenine-Möller, et. al. (2018). Real-time rendering. CRC Press (4th ed.).

FURTHER JAVA

Aims

The goal of this course is to provide students with the ability to understand the advanced programming features available in the Java programming language, completing the coverage of the language started in the Programming in Java course. The course is designed to accommodate students with diverse programming backgrounds; consequently Java is taught from first principles in a practical class setting where students can work at their own pace from a course handbook.

Objectives

At the end of the course students should

understand different mechanisms for communication between distributed applications and be able to evaluate their trade-offs;
be able to use Java generics and annotations to improve software usability, readability and safety;
understand and be able to exploit the Java class-loading mechansim;
understand and be able to use concurrency control correctly;
be able to implement a vector clock algorithm and the happens-before relation.

Recommended reading

* Goetz, B. (2006). Java concurrency in practice. Addison-Wesley.
Gosling, J., Joy, B., Steele, G., Bracha, G. and Buckley, A. (2014). The Java language specification, Java SE 8 Edition. Addison-Wesley.
http://docs.oracle.com/javase/specs/jls/se8/html/

GROUP PROJECTS
Example Risk Report
The risk report should focus on the risks to your project succeeding on time (and not, say, on general health and safety points). It is only 50 words. Example:

"Identified risks:
 * Training data requires access authorisation that has not been approved, and we don’t know if client can do this
 * Two group members have unreliable network connections and may miss meetings
 * One group member is in timezone UTC+8, and will have to attend meetings late at night
 * Bug reports on Github for a library we plan to use suggest maintenance updates will be necessary for recent Android release"

INTRODUCTION TO COMPUTER ARCHITECTURE

Aims
The aims of this course are to introduce a hardware description language (SystemVerilog) and computer architecture concepts in order to design computer systems. The parallel ECAD+Arch practical classes will allow students to apply the concepts taught in lectures.

Lectures
Part 1 - Gates to processors

Technology trends and design challenges. Current technology, technology trends, ECAD trends, challenges. [1 lecture]
Digital system design. Practicalities of mapping SystemVerilog descriptions of hardware (including a processor) onto an FPGA board. Tips and pitfalls when generating larger modular designs. [1 lecture]
Eight great ideas in computer architecture. [1 lecture]
Reduced instruction set computers and RISC-V. Introduction to the RISC-V processor design. [1 lecture]
Executable and synthesisable models. [1 lecture]
Pipelining. [2 lectures]
Memory hierarchy and caching. Caching, etc. [1 lecture]
Support for operating systems. Memory protection, exceptions, interrupts, etc. [1 lecture]
Other instruction set architectures. CISC, stack, accumulator. [1 lecture]
Part 2

Overview of Systems-on-Chip (SoCs) and DRAM. [1 lecture] High-level SoCs, DRAM storage and accessing.
Multicore Processors. [2 lectures] Communication, cache coherence, barriers and synchronisation primitives.
Graphics processing units (GPUs) [2 lectures] Basic GPU architecture and programming.
Future Directions [1 lecture] Where is computer architecture heading?
Objectives
At the end of the course students should

be able to read assembler given a guide to the instruction set and be able to write short pieces of assembler if given an instruction set or asked to invent an instruction set;
understand the differences between RISC and CISC assembler;
understand what facilities a processor provides to support operating systems, from memory management to software interrupts;
understand memory hierarchy including different cache structures and coherency needed for multicore systems;
understand how to implement a processor in SystemVerilog;
appreciate the use of pipelining in processor design;
have an appreciation of control structures used in processor design;

have an appreciation of system-on-chips and their components;
understand how DRAM stores data;
understand how multicore processors communicate;
understand how GPUs work and have an appreciation of how to program them.
Recommended reading
* Patterson, D.A. and Hennessy, J.L. (2017). Computer organization and design: The hardware/software interface RISC-V edition. Morgan Kaufmann. ISBN 978-0-12-812275-4.

Recommended further reading:

Harris, D.M. and Harris, S.L. (2012). Digital design and computer architecture. Morgan Kaufmann. ISBN 978-0-12-394424-5.
Hennessy, J. and Patterson, D. (2006). Computer architecture: a quantitative approach. Elsevier (4th ed.). ISBN 978-0-12-370490-0. (Older versions of the book are also still generally relevant.)

Pointers to sources of more specialist information are included in the lecture notes and on the associated course web page

PROGRAMMING IN C AND C++

Aims
This course aims

to provide a solid introduction to programming in C and to provide an overview of the principles and constraints that affect the way in which the C programming language has been designed and is used;
to introduce the key additional features of C++.

Lectures
Introduction to the C language. Background and goals of C. Types, expressions, control flow and strings.
(continued) Functions. Multiple compilation units. Scope. Segments. Incremental compilation. Preprocessor.
(continued) Pointers and pointer arithmetic. Function pointers. Structures and Unions.
(continued) Const and volatile qualifiers. Typedefs. Standard input/output. Heap allocation. Miscellaneous Features, Hints and Tips.
C semantics and tools. Undefined vs implementation-defined behaviour. Buffer and integer overflows. ASan, MSan, UBsan, Valgrind checkers.
Memory allocation, data structures and aliasing. Malloc/free, tree and DAG examples and their deallocation using a memory arena.
Further memory management. Reference Counting and Garbage Collection
Memory hierarchy and cache optimization. Data structure layouts. Intrusive lists. Array-of-structs vs struct-of-arrays representations. Loop blocking.
Debugging. Using print statements, assertions and an interactive debugger. Unit testing and regression.
Introduction to C++. Goals of C++. Differences between C and C++. References versus pointers. Overloaded functions.
Objects in C++. Classes and structs. Destructors. Resource Acquisition is Initialisation. Operator overloading. Virtual functions. Casts. Multiple inheritance. Virtual base classes.
Other C++ concepts. Templates and meta-programming.
Objectives
At the end of the course students should

be able to read and write C programs;
understand the interaction between C programs and the host operating system;
be familiar with the structure of C program execution in machine memory;
understand the potential dangers of writing programs in C;
understand the object model and main additional features of C++.
Recommended reading
* Kernighan, B.W. and Ritchie, D.M. (1988). The C programming language. Prentice Hall (2nd ed.).

SEMANTICS OF PROGRAMMING LANGUAGES

Aims
The aim of this course is to introduce the structural, operational approach to programming language semantics. It will show how to specify the meaning of typical programming language constructs, in the context of language design, and how to reason formally about semantic properties of programs.

Lectures
Introduction. Transition systems. The idea of structural operational semantics. Transition semantics of a simple imperative language. Language design options. [2 lectures]
Types. Introduction to formal type systems. Typing for the simple imperative language. Statements of desirable properties. [2 lectures]
Induction. Review of mathematical induction. Abstract syntax trees and structural induction. Rule-based inductive definitions and proofs. Proofs of type safety properties. [2 lectures]
Functions. Call-by-name and call-by-value function application, semantics and typing. Local recursive definitions. [2 lectures]
Data. Semantics and typing for products, sums, records, references. [1 lecture]
Subtyping. Record subtyping and simple object encoding. [1 lecture]
Semantic equivalence. Semantic equivalence of phrases in a simple imperative language, including the congruence property. Examples of equivalence and non-equivalence. [1 lecture]
Concurrency. Shared variable interleaving. Semantics for simple mutexes; a serializability property. [1 lecture]
Objectives
At the end of the course students should

be familiar with rule-based presentations of the operational semantics and type systems for some simple imperative, functional and interactive program constructs;
be able to prove properties of an operational semantics using various forms of induction (mathematical, structural, and rule-based);
be familiar with some operationally-based notions of semantic equivalence of program phrases and their basic properties.
Recommended reading
* Pierce, B.C. (2002). Types and programming languages. MIT Press.
Hennessy, M. (1990). The semantics of programming languages. Wiley. Out of print, but available on the web at
http://www.cs.tcd.ie/matthew.hennessy/splexternal2015/resources/sembookWiley.pdf
Winskel, G. (1993). The formal semantics of programming languages. MIT Press.

UNIX TOOLS

Aims

This video-lecture course gives students who have already basic Unix/Linux experience some additional practical software-engineering knowledge: how to use the shell and related tools as an efficient working environment, how to automate routine tasks, and how version-control and automated-build tools can help to avoid confusion and accidents, especially when working in teams. These are essential skills, both in industrial software development and student projects.

Lectures

Unix concepts. Brief review of Unix history and design philosophy, documentation, terminals, inter-process communication mechanisms and conventions, shell, command-line arguments, environment variables, file descriptors.
Shell concepts. Program invocation, redirection, pipes, file-system navigation, argument expansion, quoting, job control, signals, process groups, variables, locale, history and alias functions, security considerations.
Scripting. Plain-text formats, executables, #!, shell control structures and functions. Startup scripts.
Text, file and networking tools. sed, grep, chmod, find, ssh, rsync, tar, zip, etc.
Version control. diff, patch, RCS, Subversion, git.
Software development tools. C compiler, linker, debugger, make.
Perl. Introduction to a powerful scripting and text-manipulation language. [2 lectures]

Objectives

At the end of the course students should

be confident in performing routine user tasks on a POSIX system, understand command-line user-interface conventions and know how to find more detailed documentation;
appreciate how simple tools can be combined to perform a large variety of tasks;
be familiar with the most common tools, file formats and configuration practices;
be able to understand, write, and maintain shell scripts and makefiles;
appreciate how using a version-control system and fully automated build processes help to maintain reproducibility and audit trails during software development;
know enough about basic development tools to be able to install, modify and debug C source code;
have understood the main concepts of, and gained initial experience in, writing Perl scripts (excluding the facilities for object-oriented programming).

Recommended reading

Robbins, A. (2005). Unix in a nutshell. O’Reilly (4th ed.).
Schwartz, R.L., Foy, B.D. and Phoenix, T. (2011). Learning Perl. O’Reilly (6th ed.).

4th Semester

COMPILER CONSTRUCTION

Aims
This course aims to cover the main concepts associated with implementing compilers for programming languages. We use a running example called SLANG (a Small LANGuage) inspired by the languages described in 1B Semantics. A toy compiler (written in ML) is provided, and students are encouraged to extend it in various ways.

Lectures
Overview of compiler structure The spectrum of interpreters and compilers; compile-time and run-time. Compilation as a sequence of translations from higher-level to lower-level intermediate languages, where each translation preserves semantics. The structure of a simple compiler: lexical analysis and syntax analysis, type checking, intermediate representations, optimisations, code generation. Overview of run-time data structures: stack and heap. Virtual machines. [1 lecture]
Lexical analysis and syntax analysis. Lexical analysis based on regular expressions and finite state automata. Using LEX-tools. How does LEX work? Parsing based on context-free grammars and push-down automata. Grammar ambiguity, left- and right-associativity and operator precedence. Using YACC-like tools. How does YACC work? LL(k) and LR(k) parsing theory. [3 lectures]
Compiler Correctness Recursive functions can be transformed into iterative functions using the Continuation-Passing Style (CPS) transformation. CPS applied to a (recursive) SLANG interpreter to derive, in a step-by-step manner, a correct stack-based compiler. [3 lectures]
Data structures, procedures/functions Representing tuples, arrays, references. Procedures and functions: calling conventions, nested structure, non-local variables. Functions as first-class values represented as closures. Simple optimisations: inline expansion, constant folding, elimination of tail recursion, peephole optimisation. [5 lectures]
Advanced topics Run-time memory management (garbage collection). Static and dynamic linking. Objects and inheritance; implementation of method dispatch. Try-catch exception mechanisms. Compiling a compiler via bootstrapping. [4 lectures]

Objectives
At the end of the course students should understand the overall structure of a compiler, and will know significant details of a number of important techniques commonly used. They will be aware of the way in which language features raise challenges for compiler builders.

Recommended reading
* Aho, A.V., Sethi, R. and Ullman, J.D. (2007). Compilers: principles, techniques and tools. Addison-Wesley (2nd ed.).
Mogensen, T. Æ. (2011). Introduction to compiler design. Springer. http://www.diku.dk/torbenm/Basics.

COMPUTATION THEORY

Aims
The aim of this course is to introduce several apparently different formalisations of the informal notion of algorithm; to show that they are equivalent; and to use them to demonstrate that there are uncomputable functions and algorithmically undecidable problems.

Lectures
Introduction: algorithmically undecidable problems. Decision problems. The informal notion of algorithm, or effective procedure. Examples of algorithmically undecidable problems. [1 lecture]
Register machines. Definition and examples; graphical notation. Register machine computable functions. Doing arithmetic with register machines. [1 lecture]
Universal register machine. Natural number encoding of pairs and lists. Coding register machine programs as numbers. Specification and implementation of a universal register machine. [2 lectures]
Undecidability of the halting problem. Statement and proof. Example of an uncomputable partial function. Decidable sets of numbers; examples of undecidable sets of numbers. [1 lecture]
Turing machines. Informal description. Definition and examples. Turing computable functions. Equivalence of register machine computability and Turing computability. The Church-Turing Thesis. [2 lectures]
Primitive and partial recursive functions. Definition and examples. Existence of a recursive, but not primitive recursive function. A partial function is partial recursive if and only if it is computable. [2 lectures]
Lambda-Calculus. Alpha and beta conversion. Normalization. Encoding data. Writing recursive functions in the lambda-calculus. The relationship between computable functions and lambda-definable functions. [3 lectures]
Objectives
At the end of the course students should

be familiar with the register machine, Turing machine and lambda-calculus models of computability;
understand the notion of coding programs as data, and of a universal machine;
be able to use diagonalisation to prove the undecidability of the Halting Problem;
understand the mathematical notion of partial recursive function and its relationship to computability.
Recommended reading
* Hopcroft, J.E., Motwani, R. and Ullman, J.D. (2001). Introduction to automata theory, languages, and computation. Addison-Wesley (2nd ed.).
* Hindley, J.R. and Seldin, J.P. (2008). Lambda-calculus and combinators, an introduction. Cambridge University Press (2nd ed.).
Cutland, N.J. (1980). Computability: an introduction to recursive function theory. Cambridge University Press.
Davis, M.D., Sigal, R. and Weyuker, E.J. (1994). Computability, complexity and languages. Academic Press (2nd ed.).

Sudkamp, T.A. (2005). Languages and machines. Addison-Wesley (3rd ed.).

COMPUTER NETWORKING

Aims
The aim of this course is to introduce key concepts and principles of computer networks. The course will use a top-down approach to study the Internet and its protocol stack. Instances of architecture, protocol, application-examples will include email, web and media-streaming. We will cover communications services (e.g., TCP/IP) required to support such network applications. The implementation and deployment of communications services in practical networks: including wired and wireless LAN environments, will be followed by a discussion of issues of network-management. Throughout the course, the Internet's architecture and protocols will be used as the primary examples to illustrate the fundamental principles of computer networking.

Lectures
Introduction. Overview of networking using the Internet as an example. LANs and WANs. OSI reference model, Internet TCP/IP Protocol Stack. Circuit-switching, packet-switching, Internet structure, networking delays and packet loss. [3 lectures]
Link layer and local area networks. Link layer services, error detection and correction, Multiple Access Protocols, link layer addressing, Ethernet, hubs and switches, Point-to-Point Protocol. [2 lectures]
Wireless and mobile networks. Wireless links and network characteristics, Wi-Fi: IEEE 802.11 wireless LANs. [1 lecture]
Network layer addressing. Network layer services, IP, IP addressing, IPv4, DHCP, NAT, ICMP, IPv6. [3 lectures]
Network layer routing. Routing and forwarding, routing algorithms, routing in the Internet, multicast. [3 lectures]
Transport layer. Service models, multiplexing/demultiplexing, connection-less transport (UDP), principles of reliable data transfer, connection-oriented transport (TCP), TCP congestion control, TCP variants. [6 lectures]
Application layer. Client/server paradigm, WWW, HTTP, Domain Name System, P2P. [1.5 lectures]
Multimedia networking. Networked multimedia applications, multimedia delivery requirements, multimedia protocols (SIP), content distribution networks. [0.5 lecture]
Objectives
At the end of the course students should

be able to analyse a communication system by separating out the different functions provided by the network;
understand that there are fundamental limits to any communications system;
understand the general principles behind multiplexing, addressing, routing, reliable transmission and other stateful protocols as well as specific examples of each;
understand what FEC is;
be able to compare communications systems in how they solve similar problems;

have an informed view of both the internal workings of the Internet and of a number of common Internet applications and protocols.
Recommended reading
* Peterson, L.L. and Davie, B.S. (2011). Computer networks: a systems approach. Morgan Kaufmann (5th ed.). ISBN 9780123850591
Kurose, J.F. and Ross, K.W. (2009). Computer networking: a top-down approach. Addison-Wesley (5th ed.).
Comer, D. and Stevens, D. (2005). Internetworking with TCP-IP, vol. 1 and 2. Prentice Hall (5th ed.).
Stevens, W.R., Fenner, B. and Rudoff, A.M. (2003). UNIX network programming, Vol.I: The sockets networking API. Prentice Hall (3rd ed.).

FURTHER HUMAN–COMPUTER INTERACTION

Aims
This aim of this course is to provide an introduction to the theoretical foundations of Human Computer Interaction, and an understanding of how these can be applied to the design of complex technologies.

Lectures
Theory driven approaches to HCI. What is a theory in HCI? Why take a theory driven approach to HCI?
Design of visual displays. Segmentation and variables of the display plane. Modes of correspondence.
Goal-oriented interaction. Using cognitive theories of planning, learning and understanding to understand user behaviour, and what they find hard.
Designing smart systems. Using statistical methods to anticipate user needs and actions with Bayesian strategies.
Designing efficient systems. Measuring and optimising human performance through quantitative experimental methods.
Designing meaningful systems. Qualitative research methods to understand social context and requirements of user experience.
Evaluating interactive system designs. Approaches to evaluation in systems research and engineering, including Part II Projects.
Designing complex systems. Worked case studies of applying the theories to a hard HCI problem. Research directions in HCI.

Objectives
At the end of the course students should be able to apply theories of human performance and cognition to system design, including selection of appropriate techniques to analyse, observe and improve the usability of a wide range of technologies.

Recommended reading
* Preece, J., Sharp, H. and Rogers, Y. (2015). Interaction design: beyond human-computer interaction. Wiley (Currently in 4th edition, but earlier editions will suffice).

Further reading:

Carroll, J.M. (ed.) (2003). HCI models, theories and frameworks: toward a multi-disciplinary science. Morgan Kaufmann.

LOGIC AND PROOF

Aims
This course will teach logic, especially the predicate calculus. It will present the basic principles and definitions, then describe a variety of different formalisms and algorithms that can be used to solve problems in logic. Putting logic into the context of Computer Science, the course will show how the programming language Prolog arises from the automatic proof method known as resolution. It will introduce topics that are important in mechanical verification, such as binary decision diagrams (BDDs), SAT solvers and modal logic.

Lectures
Introduction to logic. Schematic statements. Interpretations and validity. Logical consequence. Inference.
Propositional logic. Basic syntax and semantics. Equivalences. Normal forms. Tautology checking using CNF.
The sequent calculus. A simple (Hilbert-style) proof system. Natural deduction systems. Sequent calculus rules. Sample proofs.
First order logic. Basic syntax. Quantifiers. Semantics (truth definition).
Formal reasoning in FOL. Free versus bound variables. Substitution. Equivalences for quantifiers. Sequent calculus rules. Examples.
Clausal proof methods. Clause form. A SAT-solving procedure. The resolution rule. Examples. Refinements.
Skolem functions, Unification and Herbrand's theorem. Prenex normal form. Skolemisation. Most general unifiers. A unification algorithm. Herbrand models and their properties.
Resolution theorem-proving and Prolog. Binary resolution. Factorisation. Example of Prolog execution. Proof by model elimination.
Satisfiability Modulo Theories. Decision problems and procedures. How SMT solvers work.
Binary decision diagrams. General concepts. Fast canonical form algorithm. Optimisations. Applications.
Modal logics. Possible worlds semantics. Truth and validity. A Hilbert-style proof system. Sequent calculus rules.
Tableaux methods. Simplifying the sequent calculus. Examples. Adding unification. Skolemisation. The world's smallest theorem prover?
Objectives
At the end of the course students should

be able to manipulate logical formulas accurately;
be able to perform proofs using the presented formal calculi;
be able to construct a small BDD;
understand the relationships among the various calculi, e.g. SAT solving, resolution and Prolog;
understand the concept of a decision procedure and the basic principles of "satisfiability modulo theories".
be able to apply the unification algorithm and to describe its uses.
Recommended reading

* Huth, M. and Ryan, M. (2004). Logic in computer science: modelling and reasoning about systems. Cambridge University Press (2nd ed.).
Ben-Ari, M. (2001). Mathematical logic for computer science. Springer (2nd ed.).

PROLOG

Aims
The aim of this course is to introduce programming in the Prolog language. Prolog encourages a different programming style to Java or ML and particular focus is placed on programming to solve real problems that are suited to this style. Practical experimentation with the language is strongly encouraged.

Lectures
Introduction to Prolog. The structure of a Prolog program and how to use the Prolog interpreter. Unification. Some simple programs.
Arithmetic and lists. Prolog's support for evaluating arithmetic expressions and lists. The space complexity of program evaluation discussed with reference to last-call optimisation.
Backtracking, cut, and negation. The cut operator for controlling backtracking. Negation as failure and its uses.
Search and cut. Prolog's search method for solving problems. Graph searching exploiting Prolog's built-in search mechanisms.
Difference structures. Difference lists: introduction and application to example programs.
Building on Prolog. How particular limitations of Prolog programs can be addressed by techniques such as Constraint Logic Programming (CLP) and tabled resolution.
Objectives
At the end of the course students should

be able to write programs in Prolog using techniques such as accumulators and difference structures;
know how to model the backtracking behaviour of program execution;
appreciate the unique perspective Prolog gives to problem solving and algorithm design;
understand how larger programs can be created using the basic programming techniques used in this course.
Recommended reading
* Bratko, I. (2001). PROLOG programming for artificial intelligence. Addison-Wesley (3rd or 4th ed.).
Sterling, L. and Shapiro, E. (1994). The art of Prolog. MIT Press (2nd ed.).

Further reading:

O'Keefe, R. (1990). The craft of Prolog. MIT Press. [This book is beyond the scope of this course, but it is very instructive. If you understand its contents, you're more than prepared for the examination.]

ARTIFICIAL INTELLIGENCE

Prerequisites: Algorithms 1, Algorithms 2. In addition the course requires some mathematics, in particular some use of vectors and some calculus. Part IA Natural Sciences Mathematics or equivalent and Discrete Mathematics are likely to be helpful although not essential. Similarly, elements of Machine Learning and Real World Data, Foundations of Data Science, Logic and Proof, Prolog and Complexity Theory are likely to be useful. This course is a prerequisite for the Part II courses Machine Learning and Bayesian Inference and Natural Language Processing.
This course is a prerequisite for: Natural Language Processing
Exam: Paper 7 Question 1, 2
Past exam questions, Moodle, timetable

Aims

The aim of this course is to provide an introduction to some fundamental issues and algorithms in artificial intelligence (AI). The course approaches AI from an algorithmic, computer science-centric perspective; relatively little reference is made to the complementary perspectives developed within psychology, neuroscience or elsewhere. The course aims to provide some fundamental tools and algorithms required to produce AI systems able to exhibit limited human-like abilities, particularly in the form of problem solving by search, game-playing, representing and reasoning with knowledge, planning, and learning.

Lectures

Introduction. Alternate ways of thinking about AI. Agents as a unifying view of AI systems. [1 lecture]
Search I. Search as a fundamental paradigm for intelligent problem-solving. Simple, uninformed search algorithms. Tree search and graph search. [1 lecture]
Search II. More sophisticated heuristic search algorithms. The A* algorithm and its properties. Improving memory efficiency: the IDA* and recursive best first search algorithms. Local search and gradient descent. [1 lecture]
Game-playing. Search in an adversarial environment. The minimax algorithm and its shortcomings. Improving minimax using alpha-beta pruning. [1 lecture]
Constraint satisfaction problems (CSPs). Standardising search problems to a common format. The backtracking algorithm for CSPs. Heuristics for improving the search for a solution. Forward checking, constraint propagation and arc consistency. [1 lecture]
Backjumping in CSPs. Backtracking, backjumping using Gaschnig’s algorithm, graph-based backjumping. [1 lecture]
Knowledge representation and reasoning I. How can we represent and deal with commonsense knowledge and other forms of knowledge? Semantic networks, frames and rules. How can we use inference in conjunction with a knowledge representation scheme to perform reasoning about the world and thereby to solve problems? Inheritance, forward and backward chaining. [1 lecture]
Knowledge representation and reasoning II. Knowledge representation and reasoning using first order logic. The frame, qualification and ramification problems. The situation calculus. [1 lecture]

Planning I. Methods for planning in advance how to solve a problem. The STRIPS language. Achieving preconditions, backtracking and fixing threats by promotion or demotion: the partial-order planning algorithm. [1 lecture]
Planning II. Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. Planning as a constraint satisfaction problem. [1 lecture]
Neural Networks I. A brief introduction to supervised learning from examples. Learning as fitting a curve to data. The perceptron. Learning by gradient descent. [1 lecture]
Neural Networks II. Multilayer perceptrons and the backpropagation algorithm. [1 lecture]
Objectives
At the end of the course students should:

appreciate the distinction between the popular view of the field and the actual research results;
appreciate the fact that the computational complexity of most AI problems requires us regularly to deal with approximate techniques;
be able to design basic problem solving methods based on AI-based search, knowledge representation, reasoning, planning, and learning algorithms.
Recommended reading
The recommended text is:

* Russell, S. and Norvig, P. (2010). Artificial intelligence: a modern approach. Prentice Hall (3rd ed.).

There are many good books available on artificial intelligence; one alternative is:

Poole, D. L. and Mackworth, A. K. (2017). Artificial intelligence: foundations of computational agents. Cambridge University Press (2nd ed.).

For some of the material you might find it useful to consult more specialised texts, in particular:

Dechter, R. (2003). Constraint processing. Morgan Kaufmann.

Cawsey, A. (1998). The essence of artificial intelligence. Prentice Hall.

Ghallab, M., Nau, D. and Traverso, P. (2004). Automated planning: theory and practice. Morgan Kaufmann.

Bishop, C.M. (2006). Pattern recognition and machine learning. Springer.

Brachman, R.J and Levesque, H.J. (2004). Knowledge Representation and Reasoning. Morgan Kaufmann.

COMPLEXITY THEORY

Aims
The aim of the course is to introduce the theory of computational complexity. The course will explain measures of the complexity of problems and of algorithms, based on time and space used on abstract models. Important complexity classes will be defined, and the notion of completeness established through a thorough study of NP-completeness. Applications to cryptography will be considered.

Syllabus
Introduction to Complexity Theory. Decidability and complexity; lower and upper bounds; the Church-Turing hypothesis.
Time complexity. Time complexity classes; polynomial time computation and the class P.
Non-determinism. Non-deterministic machines; the class NP; the problem 3SAT.
NP-completeness. Reductions and completeness; the Cook-Levin theorem; graph-theoretic problems.
More NP-complete problems. 3-colourability; scheduling, matching, set covering, and knapsack.
coNP. Complement classes. NP ∩ coNP; primality and factorisation.
Cryptography. One-way functions; the class UP.
Space complexity. Space complexity classes; Savitch's theorem.
Hierarchy theorems. Time and space hierarchy theorems.
Randomised algorithms. The class BPP; derandomisation; error-correcting codes, communication complexity.
Quantum Complexity. The classes BQP and QMA.
Interactive Proofs. IP=PSPACE; Zero Knowledge Proofs; Probabilistically Checkable Proofs (PCPs).

Objectives
At the end of the course students should

be able to analyse practical problems and classify them according to their complexity;
be familiar with the phenomenon of NP-completeness, and be able to identify problems that are NP-complete;
be aware of a variety of complexity classes and their interrelationships;
understand the role of complexity analysis in cryptography.
Recommended reading
A Survey of P vs. NP by Scott Aaronson.
Computational Complexity: A Modern Approach by Arora and Barak
Computational Complexity: A Conceptual Perspective by Oded Goldreich
Mathematics and Computation by Avi Wigderson

CYBERSECURITY

Aims

In today's digital society, computer security is vital for commercial competitiveness, national security and privacy of individuals. Protection against both criminal and state-sponsored attacks will need a large cohort of skilled individuals with an understanding of the principles of security and with practical experience of the application of these principles. We want you to become one of them. In this adversarial game, to defeat the bad guys, the good guys have to be at least as skilled at them. Therefore this course has a strong foundation of becoming proficient at actual attacks. A theoretical understanding is of course necessary, but without some practice the bad guys will run circles around the good guys. In 12 hours I can't bring you from zero to the stage where you can invent new attacks and countermeasures, but at least by practicing and dissecting common attacks (akin to "studying the classics") I'll give you a feeling for the kind of adversarial thinking that is required in this field. Large portions of this course are hands-on: you will need to acquire the relevant skills rather than just reading about it. The recommended course textbook will be especially helpful to those with no prior low-level hacking experience.

Lectures

Introduction.
The adversarial nature of security; thinking like the attacker; confidentiality, integrity and availability; systems-level thinking; Trusted Computing Base; role of cryptography.

Fundamentals of access control (chapter 1).
Discretionary vs mandatory access control; discretionary access control in POSIX; file permissions, file ownership, groups, permission bits.

Software security (spanning 4 book chapters)
Setuid (chapter 2): privileged programs, attack surfaces, exploitable vulnerabilities, privilege escalation from regular user to root.
Buffer overflow (chapter 4): stack memory layout, exploiting a buffer overflow vulnerability, guessing unknown addresses, payload, countermeasures and counter-countermeasures.
Return to libc and return-oriented programming (chapter 5): exploiting a non-executable stack, chaining function calls, chaining ROP gadgets.
SQL injection (chapter 14): vulnerability and its exploitation, countermeasures, input filtering, prepared statements.

Authentication.
Something you know, have, are; passwords, their dominance, their shortcomings and the many attempts at replacing them; password cracking; tokens; biometrics; taxonomy of Single Sign-On systems; password managers.

Web and internet security (spanning 3 book chapters).
Cross-Site Request Forgery (chapter 12): why cross-site requests, CSRF attacks on HTTP GET and HTTP POST, countermeasures.

Cross-Site Scripting (chapter 13): non-persistent vs persistent XSS, javascript, self-propagating XSS worm, countermeasures;

Human factors.
Users are not the enemy; Compliance budget; Prospect theory; 7 principles for systems security.

Additional topics.
Viruses, worms and quines; lockpicking; privilege escalation in physical locks; conclusions.

To complete the course successfully, students must acquire the practical ability to carry out (as opposed to just describing) the exploits mentioned in the syllabus, given a vulnerable system. The low-level hands-on portions of the course are supported by the very helpful course textbook. Those who choose not to study on the recommended textbook will be severely disadvantaged.

Recommended textbook: Wenliang Du. Computer Security: A Hands-on Approach. 3rd Edition. ISBN: 978-17330039-5-7. Published: 1 May 2022.

https://www.handsonsecurity.net. Some chapters freely available online.

Optional additional books (not substitutes):

Ross Anderson, Security Engineering 3rd Ed, Wiley, 2020. ISBN: 978-1-119-64278-7. https://www.cl.cam.ac.uk/~rja14/book.html. Some chapters freely available online.

Paul van Oorschot, Computer Security and the Internet, Springer, 2020. ISBN: 978-3-030-33648-6 (hardcopy), 978-3-030-33649-3 (eBook). https://people.scs.carleton.ca/~paulv/toolsjewels.html. All chapters freely available online.

FORMAL MODELS OF LANGUAGE

Aims
This course studies formal models of language and considers how they might be relevant to the processing and acquisition of natural languages. The course will extend knowledge of formal language theory; introduce several new grammars; and use concepts from information theory to describe natural language.

Lectures
Natural language and the Chomsky hierarchy 1. Recap classes of language. Closure properties of language classes. Recap pumping lemma for regular languages. Discussion of relevance (or not) to natural languages (example embedded clauses in English).
Natural language and the Chomsky hierarchy 2. Pumping lemma for context free languages. Discussion of relevance (or not) to natural languages (example Swiss-German cross serial dependancies). Properties of minimally context sensitive languages. Introduction to tree adjoining grammars.
Language processing and context free grammar parsing 1. Recap of context free grammar parsing. Language processing predictions based on top down parsing models (example Yngve's language processing predictions). Language processing predictions based on probabilistic parsing (example Halle's language processing predictions).
Language processing and context free grammar parsing 2. Introduction to context free grammar equivalent dependancy grammars. Language processing predictions based on Shift-Reduce parsing (examples prosodic look-ahead parsers, Parsey McParseface).
Grammar induction of language classes. Introduction to grammar induction. Discussion of relevance (or not) to natural language acquisition. Gold's theorem. Introduction to context free grammar equivalent categorial grammars and their learnable classes.
Natural language and information theory 1. Entropy and natural language typology. Uniform information density as a predictor for language processing.
Natural language and information theory 2. Noisy channel encoding as a model for spelling error, translation and language processing.
Vector space models and word vectors. Introduction to word vectors (example Word2Vec). Word vectors as predictors for semantic language processing.
Objectives
At the end of the course students should

understand how known natural languages relate to formal languages in the Chomsky hierarchy;
have knowledge of several context free grammars equivalents;
understand how we might make predictions about language processing and language acquisition from formal models;
know how to use information theoretic concepts to describe aspects of natural language.
Recommended reading
* Jurafsky, D. and Martin, J. (2008). Speech and language processing. Prentice Hall.
Manning, C.D. and Schutze, H. (1999) Foundations of statistical natural language processing. MIT Press.

Ruslan, M. (2003) The Oxford handbook of computational linguistics. Oxford University Press.
Clark, A., Fox, C. and Lappin, S. (2010) The handbook of computational linguistics and natural language processing. Wiley-Blackwell.
Kozen, D. (1997) Automata and computibility. Springer.

5th Semester

BIOINFORMATICS

Aims

This course focuses on algorithms used in Bioinformatics and System Biology. Most of the algorithms are general and can be applied in other fields on multidimensional and noisy data. All the necessary biological terms and concepts useful for the course and the examination will be given in the lectures. The most important software implementing the described algorithms will be demonstrated.

Lectures

Introduction to biological data: Bioinformatics as an interesting field in computer science. Computing and storing information with DNA (including Adleman's experiment).
Dynamic programming. Longest common subsequence, DNA global and local alignment, linear space alignment, Nussinov algorithm for RNA, heuristics for multiple alignment. (Vol. 1, chapter 5)
Sequence database search. Blast. (see notes and textbooks)
Genome sequencing. De Bruijn graph. (Vol. 1, chapter 3)
Phylogeny. Distance based algorithms (UPGMA, Neighbour-Joining). Parsimony-based algorithms. Examples in Computer Science. (Vol. 2, chapter 7)
Clustering. Hard and soft K-means clustering, use of Expectation Maximization in clustering, Hierarchical clustering, Markov clustering algorithm. (Vol. 2, chapter 8)
Genomics Pattern Matching. Suffix Tree String Compression and the Burrows-Wheeler Transform. (Vol. 2, chapter 9)
Hidden Markov Models. The Viterbi algorithm, profile HMMs for sequence alignment, classifying proteins with profile HMMs, soft decoding problem, Baum-Welch learning. (Vol. 2, chapter 10)

Objectives

At the end of this course students should

understand Bioinformatics terminology;
have mastered the most important algorithms in the field;
be able to work with bioinformaticians and biologists;
be able to find data and literature in repositories.

Recommended reading

* Compeau, P. and Pevzner, P.A. (2015). Bioinformatics algorithms: an active learning approach. Active Learning Publishers.
Durbin, R., Eddy, S., Krough, A. and Mitchison, G. (1998). Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press.
Jones, N.C. and Pevzner, P.A. (2004). An introduction to bioinformatics algorithms. MIT Press.
Felsenstein, J. (2003). Inferring phylogenies. Sinauer Associates.

BUSINESS STUDIES

Aims
How to start and run a computer company; the aims of this course are to introduce students to all the things that go to making a successful project or product other than just the programming. The course will survey some of the issues that students are likely to encounter in the world of commerce and that need to be considered when setting up a new computer company.

See also Business Seminars in the Easter Term.

Lectures
So you’ve got an idea? Introduction. Why are you doing it and what is it? Types of company. Market analysis. The business plan.
Money and tools for its management. Introduction to accounting: profit and loss, cash flow, balance sheet, budgets. Sources of finance. Stocks and shares. Options and futures.
Setting up: legal aspects. Company formation. Brief introduction to business law; duties of directors. Shares, stock options, profit share schemes and the like. Intellectual Property Rights, patents, trademarks and copyright. Company culture and management theory.
People. Motivating factors. Groups and teams. Ego. Hiring and firing: employment law. Interviews. Meeting techniques.
Project planning and management. Role of a manager. PERT and GANTT charts, and critical path analysis. Estimation techniques. Monitoring.
Quality, maintenance and documentation. Development cycle. Productization. Plan for quality. Plan for maintenance. Plan for documentation.
Marketing and selling. Sales and marketing are different. Marketing; channels; marketing communications. Stages in selling. Control and commissions.
Growth and exit routes. New markets: horizontal and vertical expansion. Problems of growth; second system effects. Management structures. Communication. Exit routes: acquisition, floatation, MBO or liquidation. Futures: some emerging ideas for new computer businesses. Summary. Conclusion: now you do it!
Objectives
At the end of the course students should

be able to write and analyse a business plan;
know how to construct PERT and GANTT diagrams and perform critical path analysis;
appreciate the differences between profitability and cash flow, and have some notion of budget estimation;
have an outline view of company formation, share structure, capital raising, growth and exit routes;
have been introduced to concepts of team formation and management;
know about quality documentation and productization processes;
understand the rudiments of marketing and the sales process.
Recommended reading

Lang, J. (2001). The high-tech entrepreneur's handbook: how to start and run a high-tech company. FT.COM/Prentice Hall.

Students will be expected to be able to use Microsoft Excel and Microsoft Project.

For additional reading on a lecture-by-lecture basis, please see the course website.

Students are strongly recommended to enter the CU Entrepreneurs Business Ideas Competition http://www.cue.org.uk/

DENOTATIONAL SEMANTICS

Aims
The aims of this course are to introduce domain theory and denotational semantics, and to show how they provide a mathematical basis for reasoning about the behaviour of programming languages.

Lectures
Introduction.The denotational approach to the semantics of programming languages.Recursively defined objects as limits of successive approximations.
Least fixed points.Complete partial orders (cpos) and least elements.Continuous functions and least fixed points.
Constructions on domains.Flat domains.Product domains. Function domains.
Scott induction.Chain-closed and admissible subsets of cpos and domains.Scott's fixed-point induction principle.
PCF.The Scott-Plotkin language PCF.Evaluation. Contextual equivalence.
Denotational semantics of PCF.Denotation of types and terms. Compositionality. Soundness with respect to evaluation. [2 lectures].
Relating denotational and operational semantics.Formal approximation relation and its fundamental property.Computational adequacy of the PCF denotational semantics with respect to evaluation. Extensionality properties of contextual equivalence. [2 lectures].
Full abstraction.Failure of full abstraction for the domain model. PCF with parallel or.
Objectives
At the end of the course students should

be familiar with basic domain theory: cpos, continuous functions, admissible subsets, least fixed points, basic constructions on domains;
be able to give denotational semantics to simple programming languages with simple types;
be able to apply denotational semantics; in particular, to understand the use of least fixed points to model recursive programs and be able to reason about least fixed points and simple recursive programs using fixed point induction;
understand the issues concerning the relation between denotational and operational semantics, adequacy and full abstraction, especially with respect to the language PCF.
Recommended reading
Winskel, G. (1993). The formal semantics of programming languages: an introduction. MIT Press.
Gunter, C. (1992). Semantics of programming languages: structures and techniques. MIT Press.
Tennent, R. (1991). Semantics of programming languages. Prentice Hall.

INFORMATION THEORY

Aims
This course introduces the principles and applications of information theory: how information is measured in terms of probability and various entropies, how these are used to calculate the capacity of communication channels, with or without noise, and to measure how much random variables reveal about each other. Coding schemes including error correcting codes are studied along with data compression, spectral analysis, transforms, and wavelet coding. Applications of information theory are reviewed, from astrophysics to pattern recognition.

Lectures
Information, probability, uncertainty, and surprise. How concepts of randomness and uncertainty are related to information. How the metrics of information are grounded in the rules of probability. Shannon Information. Weighing problems and other examples.
Entropy of discrete variables. Definition and link to Shannon information. Joint entropy, Mutual information. Visual depictions of the relationships between entropy types. Why entropy gives fundamental measures of information content.
Source coding theorem and data compression; prefix, variable-, and fixed-length codes. Information rates; Asymptotic equipartition principle; Symbol codes; Huffman codes and the prefix property. Binary symmetric channels. Capacity of a noiseless discrete channel. Stream codes.
Noisy discrete channel coding. Joint distributions; mutual information; Conditional Entropy; Error-correcting codes; Capacity of a discrete channel. Noisy channel coding theorem.
Entropy of continuous variables. Differential entropy; Mutual information; Channel Capacity; Gaussian channels.
Entropy for comparing probability distributions and for machine learning. Relative entropy/KL divergence; cross-entropy; use as loss function.
Applications of information theory in other sciences.
Objectives
At the end of the course students should be able to

calculate the information content of a random variable from its probability distribution;
relate the joint, conditional, and marginal entropies of variables in terms of their coupled probabilities;
define channel capacities and properties using Shannon's Theorems;
construct efficient codes for data on imperfect communication channels;
generalize the discrete concepts to continuous signals on continuous channels;
understand encoding and communication schemes in terms of the spectral properties of signals and channels;
describe compression schemes, and efficient coding using wavelets and other representations for data.
Recommended reading
Mackay's Information Theory, inference and Learning Algorithms

Stone's Information Theory: A Tutorial Introduction.

LATEX AND JULIA

Aims
Introduction to two widely-used languages for typesetting dissertations and scientific publications, for prototyping numerical algorithms and to visualize results.

Lectures
LATEX. Workflow example, syntax, typesetting conventions, non-ASCII characters, document structure, packages, mathematical typesetting, graphics and figures, cross references, build tools.
Julia. Tools for technical computing and visualization. The Array type and its operators, 2D/3D plotting, functions and methods, notebooks, packages, vectorized audio demonstration.
Objectives
Students should be able to avoid the most common LATEX mistakes, to prototype simple image and signal-processing algorithms in Julia, and to visualize the results.

Recommended reading
* Lamport, L. (1994). LATEX – a documentation preparation system user's guide and reference manual. Addison-Wesley (2nd ed.).
Mittelbach, F., et al. (2023). The LATEX companion. Addison-Wesley (3rd ed.).

PRINCIPLES OF COMMUNICATIONS

Aims
This course aims to provide a detailed understanding of the underlying principles for how communications systems operate. Practical examples (from wired and wireless communications, the Internet, and other communications systems) are used to illustrate the principles.

Lectures
Introduction. Course overview. Abstraction, layering. Review of structure of real networks, links, end systems and switching systems. [1 lecture]
Routing. Central versus Distributed Routing Policy Routing. Multicast Routing Circuit Routing [6 lectures]
Flow control and resource optimisation. 1[Control theory] is a branch of engineering familiar to people building dynamic machines. It can be applied to network traffic. Stemming the flood, at source, sink, or in between? Optimisation as a model of networkand user. TCP in the wild. [3 lectures]
Packet Scheduling. Design choices for scheduling and queue management algorithms for packet forwarding, and fairness. [2 lectures]
The big picture for managing traffic. Economics and policy are relevant to networks in many ways. Optimisation and game theory are both relevant topics discussed here. [2 lectures]
System Structures and Summary. Abstraction, layering. The structure of real networks, links, end systems and switching. [2 lectures]
1 Control theory was not taught and will not be the subject of any exam question.

Objectives
At the end of the course students should be able to explain the underlying design and behaviour of protocols and networks, including capacity, topology, control and use. Several specific mathematical approaches are covered (control theory, optimisation).

Recommended reading
* Keshav, S. (2012). Mathematical Foundations of Computer Networking. Addison Wesley. ISBN 9780321792105
Background reading:
Keshav, S. (1997). An engineering approach to computer networking. Addison-Wesley (1st ed.). ISBN 0201634422
Stevens, W.R. (1994). TCP/IP illustrated, vol. 1: the protocols. Addison-Wesley (1st ed.). ISBN 0201633469

TYPES

Aims
The aim of this course is to show by example how type systems for programming languages can be defined and their properties developed, using techniques that were introduced in the Part IB course on Semantics of Programming Languages. The emphasis is on type systems for functional languages and their connection to constructive logic.

Lectures
Introduction. The role of type systems in programming languages. Review of rule-based formalisation of type systems. [1 lecture]
Propositions as types. The Curry-Howard correspondence between intuitionistic propositional calculus and simply-typed lambda calculus. Inductive types and iteration. Consistency and termination. [2 lectures]
Polymorphic lambda calculus (PLC). PLC syntax and reduction semantics. Examples of datatypes definable in the polymorphic lambda calculus. Type inference. [3 lectures]
Monads and effects. Explicit versus implicit effects. Using monadic types to control effects. References and polymorphism. Recursion and looping. [2 lectures]
Continuations and classical logic. First-class continuations and control operators. Continuations as Curry-Howard for classical logic. Continuation-passing style. [2 lectures]
Dependent types. Dependent function types. Indexed datatypes. Equality types and combining proofs with programming. [2 lectures]
Objectives
At the end of the course students should

be able to use a rule-based specification of a type system to carry out type checking and type inference;
understand by example the Curry-Howard correspondence between type systems and logics;
understand how types can be used to control side-effects in programming;
appreciate the expressive power of parametric polymorphism and dependent types.
Recommended reading
* Pierce, B.C. (2002). Types and programming languages. MIT Press.
Pierce, B. C. (Ed.) (2005). Advanced Topics in Types and Programming Languages. MIT Press.
Girard, J-Y. (tr. Taylor, P. and Lafont, Y.) (1989). Proofs and types. Cambridge University Press.

ADVANCED DATA SCIENCE

Practicals
The module will have a practical session for each of the three stages of the pipeline. Each of the parts will be tied into an aspect of the final project.

Objectives
At the end of the course students will be familiar with the purpose of data science, how it differs from the closely related fields of machine learning, statistics and artificial intelligence and what a typical data analysis pipeline looks like in practice. As well as emphasising the importance of analysis methods we will introduce a formalism for organising how data science is done in practice and what the different aspects the data scientist faces when giving data-driven answers to questions of interest.

Recommended reading
Simon Rogers et al. (2016). A First Course in Machine Learning, Second Edition. [] Chapman and Hall/CRC, nil
Shai Shalev-Shwartz et al. (2014). Understanding Machine Learning: From Theory to Algorithms. New York, NY, USA: Cambridge University Press
Christopher M. Bishop (2006). Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc.
Assessment
Practical There will be four practicals contributing 20% to the final module mark. Data science is a topic where there is rarely a single
correct answer therefore the important thing is that an informed attempt has been made to address the questions that can be supported and motivated.
Report The course will be concluded by an individual report covering the material in the course through
practical exercise working with real data. This report will make up 80% of the mark for the course.

AFFECTIVE ARTIFICIAL INTELLIGENCE

Synopsis

Affective Artificial Intelligence (Affective AI) aims to imbue machines with social and emotional intelligence (EQ). More specifically, Affective AI aims to create artificially intelligent systems and machines that can recognize, interpret, process, and simulate human social signals and behaviours, expressions, and emotions, to enhance human-AI interaction and communication. \r\n\r\nTo achieve this goal, Affective AI draws upon various scientific disciplines, including machine learning, computer vision, speech / natural language / signal processing, psychology and cognitive science, and ethics and social sciences.

Background & Aims

Affective Artificial Intelligence (Affective AI) aims to imbue machines with social and emotional intelligence (EQ). More specifically, Affective AI aims to create artificially intelligent systems and machines that can recognize, interpret, process, and simulate human social signals and behaviours, expressions, and emotions, to enhance human-AI interaction and communication.

To achieve this goal, Affective AI draws upon various scientific disciplines, including machine learning, computer vision, speech / natural language / signal processing, psychology and cognitive science, and ethics and social sciences.

Affective AI has direct applications in and implications for the design of innovative interactive technology (e.g., interaction with chat bots, virtual agents, robots), single and multi-user smart environments (e.g., in-car/ virtual / augmented / mixed reality, serious games), public speaking and cognitive training, and clinical and biomedical studies (e.g., autism, depression, pain).

The aim of this module is to impart knowledge and ability needed to make informed choices of models, data, and machine learning techniques for sensing, recognition, and generation of affective and social behaviour (e.g., smile, frown, head nodding/shaking, agreement/disagreement) in order to create Affectively intelligent AI systems, with a consideration for various ethical issues (e.g., privacy, bias) arising from the real-world deployment of these systems.

Syllabus

The following list provides a representative list of topics:

Introduction, definitions, and overview
Theories from various disciplines
Sensing from multiple modalities (e.g., vision, audio, bio signals, text)
Data acquisition and annotation
Signal processing / feature extraction

Learning / prediction / recognition and evaluation
Behaviour synthesis / generation (e.g., for embodied agents / robots)
Advanced topics and ethical considerations (e.g., bias and fairness)
Cross-disciplinary applications (via seminar presentations and discussions)
Guest lectures (diverse topics – changes each year)
Hands-on research and programming work (i.e., mini project & report)

Objectives
On completion of this module, students will:

understand and demonstrate knowledge in key characteristics of affectively intelligent AI, which include:
Recognition: How to equip Affective AI systems with capabilities of analysing facial expressions, vocal intonations, gestures, and other physiological signals to infer human affective states?
Generation: How to enable affectively intelligent AI simulate expressions of emotions in machines, allowing them to respond in a more human-like manner, such as virtual agents and humanoid robots, showing empathy or sympathy?
Adaptation and Personalization: How to enable affectively intelligent AI adapt system responses based on users' affective states and/or needs, or tailor interactions and experiences to individual users based on their expressivity / emotional profiles, personalities, and past interactions?
Empathetic Communication: How to design Affective AI systems to communicate with users in a way that demonstrates empathy, understanding, and sensitivity to their affective states and needs?
Ethical and societal considerations: What are the various human differences, ethical guidelines, and societal impacts that need to be considered when designing and deploying Affective AI to ensure that these systems respect users' privacy, autonomy, and well-being?
comprehend and apply (appropriate) methods for collection, analysis, representation, and evaluation of human affective and communicative behavioural data.
enhance programming skills for creating and implementing (components of) Affective AI systems.
demonstrate critical thinking, analysis and synthesis while deciding on 'when' and 'how' to incorporate human affect and social signals in a specific AI system context and gain practical experience in proposing and justifying computational solution(s) of suitable nature and scope.

Assessment - Part II Students
Seminar presentation: 20%
Participating in Q&A and discussions: 10%
Mid-term mini project report and presentation (as a team of 2): 10%
Final mini project report & code (as a team of 2): 60%

CATEGORY THEORY

Aims
Category theory provides a unified treatment of mathematical properties and constructions that can be expressed in terms of 'morphisms' between structures. It gives a precise framework for comparing one branch of mathematics (organized as a category) with another and for the transfer of problems in one area to another. Since its origins in the 1940s motivated by connections between algebra and geometry, category theory has been applied to diverse fields, including computer science, logic and linguistics. This course introduces the basic notions of category theory: adjunction, natural transformation, functor and category. We will use category theory to organize and develop the kinds of structure that arise in models and semantics for logics and programming languages.

Syllabus
Introduction; some history. Definition of category. The category of sets and functions. Commutative diagrams. Examples of categories: preorders and monotone functions; monoids and monoid homomorphisms; a preorder as a category; a monoid as a category. Definition of isomorphism. Informal notion of a 'category-theoretic' property.
Terminal objects. The opposite of a category and the duality principle. Initial objects. Free monoids as initial objects.
Binary products and coproducts. Cartesian categories.
Exponential objects: in the category of sets and in general. Cartesian closed categories: definition and examples.
Intuitionistic Propositional Logic (IPL) in Natural Deduction style. Semantics of IPL in a cartesian closed preorder.
Simply Typed Lambda Calculus (STLC). The typing relation. Semantics of STLC types and terms in a cartesian closed category (ccc). The internal language of a ccc. The Curry-Howard-Lawvere correspondence.
Functors. Contravariance. Identity and composition for functors.
Size: small categories and locally small categories. The category of small categories. Finite products of categories.
Natural transformations. Functor categories. The category of small categories is cartesian closed.
Hom functors. Natural isomorphisms. Adjunctions. Examples of adjoint functors. Theorem characterising the existence of right (respectively left) adjoints in terms of a universal property.
Dependent types. Dependent product sets and dependent function sets as adjoint functors.
Equivalence of categories. Example: the category of I-indexed sets and functions is equivalent to the slice category Set/I.
Presheaves. The Yoneda Lemma. Categories of presheaves are cartesian closed.
Monads. Modelling notions of computation as monads. Moggi's computational lambda calculus.
Objectives
On completion of this module, students should:

be familiar with some of the basic notions of category theory and its connections with logic and programming language semantics
Assessment
a graded exercise sheet (25% of the final mark), and
a take-home test (75%)
Recommended reading
Awodey, S. (2010). Category theory. Oxford University Press (2nd ed.).

Crole, R. L. (1994). Categories for types. Cambridge University Press.

Lambek, J. and Scott, P. J. (1986). Introduction to higher order categorical logic. Cambridge University Press.

Pitts, A. M. (2000). Categorical Logic. Chapter 2 of S. Abramsky, D. M. Gabbay and T. S. E. Maibaum (Eds) Handbook of Logic in Computer Science, Volume 5. Oxford University Press. (Draft copy available here.)

Class Size
This module can accommodate upto 15 Part II students plus 15 MPhil / Part III students.

DIGITAL SIGNAL PROCESSING

Aims

This course teaches basic signal-processing principles necessary to understand many modern high-tech systems, with examples from audio processing, image coding, radio communication, radar, and software-defined radio. Students will gain practical experience from numerical experiments in programming assignments (in Julia, MATLAB or NumPy).

Lectures

Signals and systems. Discrete sequences and systems: types and properties. Amplitude, phase, frequency, modulation, decibels, root-mean square. Linear time-invariant systems, convolution. Some examples from electronics, optics and acoustics.
Phasors. Eigen functions of linear time-invariant systems. Review of complex arithmetic. Phasors as orthogonal base functions.
Fourier transform. Forms and properties of the Fourier transform. Convolution theorem. Rect and sinc.
Dirac's delta function. Fourier representation of sine waves, impulse combs in the time and frequency domain. Amplitude-modulation in the frequency domain.
Discrete sequences and spectra. Sampling of continuous signals, periodic signals, aliasing, interpolation, sampling and reconstruction, sample-rate conversion, oversampling, spectral inversion.
Discrete Fourier transform. Continuous versus discrete Fourier transform, symmetry, linearity, FFT, real-valued FFT, FFT-based convolution, zero padding, FFT-based resampling, deconvolution exercise.
Spectral estimation. Short-time Fourier transform, leakage and scalloping phenomena, windowing, zero padding. Audio and voice examples. DTFM exercise.
Finite impulse-response filters. Properties of filters, implementation forms, window-based FIR design, use of frequency-inversion to obtain high-pass filters, use of modulation to obtain band-pass filters.
Infinite impulse-response filters. Sequences as polynomials, z-transform, zeros and poles, some analog IIR design techniques (Butterworth, Chebyshev I/II, elliptic filters, second-order cascade form).
Band-pass signals. Band-pass sampling and reconstruction, IQ up and down conversion, superheterodyne receivers, software-defined radio front-ends, IQ representation of AM and FM signals and their demodulation.
Digital communication. Pulse-amplitude modulation. Matched-filter detector. Pulse shapes, inter-symbol interference, equalization. IQ representation of ASK, BSK, PSK, QAM and FSK signals. [2 hours]
Random sequences and noise. Random variables, stationary and ergodic processes, autocorrelation, cross-correlation, deterministic cross-correlation sequences, filtered random sequences, white noise, periodic averaging.
Correlation coding. Entropy, delta coding, linear prediction, dependence versus correlation, random vectors, covariance, decorrelation, matrix diagonalization, eigen decomposition,

Karhunen-Loève transform, principal component analysis. Relation to orthogonal transform coding using fixed basis vectors, such as DCT.
Lossy versus lossless compression. What information is discarded by human senses and can be eliminated by encoders? Perceptual scales, audio masking, spatial resolution, colour coordinates, some demonstration experiments.
Quantization, image coding standards. Uniform and logarithmic quantization, A/μ-law coding, dithering, JPEG.
Objectives
apply basic properties of time-invariant linear systems;
understand sampling, aliasing, convolution, filtering, the pitfalls of spectral estimation;
explain the above in time and frequency domain representations;
use filter-design software;
visualize and discuss digital filters in the z-domain;
use the FFT for convolution, deconvolution, filtering;
implement, apply and evaluate simple DSP applications;
familiarity with a number of signal-processing concepts used in digital communication systems
Recommended reading
Lyons, R.G. (2010). Understanding digital signal processing. Prentice Hall (3rd ed.).
Oppenheim, A.V. and Schafer, R.W. (2007). Discrete-time digital signal processing. Prentice Hall (3rd ed.).
Stein, J. (2000). Digital signal processing – a computer science perspective. Wiley.

Class size
This module can accommodate a maximum of 24 students (16 Part II students and 8 MPhil students)

Assessment - Part II students
Three homework programming assignments, each comprising 20% of the mark
Written test, comprising 40% of the total mark.

MACHINE VISUAL PERCEPTION

Aims

This course aims at introducing the theoretical foundations and practical techniques for machine perception, the capability of computers to interpret data resulting from sensor measurements. The course will teach the fundamentals and modern techniques of machine perception, i.e. reconstructing the real world starting from sensor measurements with a focus on machine perception for visual data. The topics covered will be image/geometry representations for machine perception, semantic segmentation, object detection and recognition, geometry capture, appearance modeling and acquisition, motion detection and estimation, human-in-the-loop machine perception, select topics in applied machine perception.

Machine perception/computer vision is a rapidly expanding area of research with real-world applications. An understanding of machine perception is also important for robotics, interactive graphics (especially AR/VR), applied machine learning, and several other fields and industries. This course will provide a fundamental understanding of and practical experience with the relevant techniques.

Learning outcomes

Students will understand the theoretical underpinnings of the modern machine perception techniques for reconstructing models of reality starting from an incomplete and imperfect view of reality.
Students will be able to apply machine perception theory to solve practical problems, e.g. classification of images, geometry capture.
Students will gain an understanding of which machine perception techniques are appropriate for different tasks and scenarios.
Students will have hands-on experience with some of these techniques via developing a functional machine perception system in their projects.
Students will have practical experience with the current prominent machine perception frameworks.

Syllabus

The fundamentals of machine learning for machine perception
Deep neural networks and frameworks for machine perception
Semantic segmentation of objects and humans
Object detection and recognition
Motion estimation, tracking and recognition
3D geometry capture
Appearance modeling and acquisition
Select topics in applied machine perception

Assessment

A practical exercise, worth 20% of the mark. This will cover the basics and theory of machine perception and some of the practical techniques the students will likely use in their projects. This is individual work. No GPU hours will be needed for the practical work.
A machine perception project worth 80% of the marks:

Course projects will be selected by the students following the possible project themes proposed by the lecturer, and will be checked by the lecturer for appropriateness.
The students will form groups of 2-3 to design, implement, report, and present a project to tackle a given task in machine perception.
The final mark will be composed of an implementation/report mark (60%) and a presentation mark (20%). Each team member will be evaluated based on her/his contribution.
Each project will have extensions to be completed only by the ACS students. Each student will write a different part of the report, whose author will be clearly marked. Each student will further summarise her/his contributions to the project in the same report.
Recommended Reading List
Computer Vision: Algorithms and Applications, Richard Szeliski, Springer, 2010.
Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, MIT Press, 2016.
Machine Learning and Visual Perception, Baochang Zhang, De Gruyter, 2020.

NATURAL LANGUAGE PROCESSING

Aims
This course introduces the fundamental techniques of natural language processing. It aims to explain the potential and the main limitations of these techniques. Some current research issues are introduced and some current and potential applications discussed and evaluated. Students will also be introduced to practical experimentation in natural language processing.

Lectures
Overview. Brief history of NLP research, some current applications, components of NLP systems.
Morphology and Finite State Techniques. Morphology in different languages, importance of morphological analysis in NLP, finite-state techniques in NLP.
Part-of-Speech Tagging and Log-Linear Models. Lexical categories, word tagging, corpora and annotations, empirical evaluation.
Phrase Structure and Structure Prediction. Phrase structures, structured prediction, context-free grammars, weights and probabilities. Some limitations of context-free grammars.
Dependency Parsing. Dependency structure, grammar-free parsing, incremental processing.
Gradient Descent and Neural Nets. Parameter optimisation by gradient descent. Non-linear functions with neural network layers. Log-linear model as softmax layer. Current findings of Neural NLP.
Word representations. Representing words with vectors, count-based and prediction-based approaches, similarity metrics.
Recurrent Neural Networks. Modelling sequences, parameter sharing in recurrent neural networks, neural language models, word prediction.
Compositional Semantics. Logical representations, compositional semantics, lambda calculus, inference and robust entailment.
Lexical Semantics. Semantic relations, WordNet, word senses.
Discourse. Discourse relations, anaphora resolution, summarization.
Natural Language Generation. Challenges of natural language generation (NLG), tasks in NLG, surface realisation.
Practical and assignments. Students will build a natural language processing system which will be trained and evaluated on supplied data. The system will be built from existing components, but students will be expected to compare approaches and some programming will be required for this. Several assignments will be set during the practicals for assessment.
Objectives
By the end of the course students should:

be able to discuss the current and likely future performance of several NLP applications;
be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing, parsing, word sense disambiguation etc.;
understand how these techniques draw on and relate to other areas of computer science.
Recommended reading

Jurafsky, D. & Martin, J. (2023). Speech and language processing. Prentice Hall (3rd ed. draft, online).

Assessment - Part II Students
Assignment 1 - 10% of marks
Assignment 2 - 60% of marks
Assignment 3 - 30% of marks

PRACTICAL RESEARCH IN HUMAN-CENTRED AI

Aims
This is an advanced course in human-computer interaction, with a specialist focus on intelligent user interfaces and interaction with machine-learning and artificial intelligence technologies. The format will be largely Practical, with students carrying out a mini-project involving empirical research investigation. These studies will investigate human interaction with some kind of model-based system for planning, decision-making, automation etc. Possible study formats might include: System evaluation, Field observation, Hypothesis testing experiment, Design intervention or Corpus analysis, following set examples from recent research publications. Project work will be formally evaluated through a report and presentation.

Lectures
(note that Lectures 2-7 also include one hour class discussion of practical work)

- Current research themes in intelligent user interfaces
- Program synthesis
- Mixed initiative interaction
- Interpretability / explainable AI
- Labelling as a fundamental problem
- Machine learning risks and bias
- Visualisation and visual analytics
- Student research presentations

Objectives
By the end of the course students should:

- be familiar with current state of the art in intelligent interactive systems
- understand the human factors that are most critical in the design of such systems
- be able to evaluate evidence for and against the utility of novel systems
- have experience of conducting user studies meeting the quality criteria of this field
- be able to write up and present user research in a professional manner

Class Size
This module can accommodate upto 20 Part II, Part III and MPhil students.

Recommended reading
Brad A. Myers and Richard McDaniel (2000). Demonstrational Interfaces: Sometimes You Need a Little Intelligence, Sometimes You Need a Lot.

Alan Blackwell (2024). Moral Codes: Designing alternatives to AI

Assessment - Part II Students
The format will be largely practical, with students carrying out an individual mini-project involving empirical research investigation.

Assignment 1: six incremental submissions which together contribute 20% to the final module mark.

Assignment 2: Final report - 80% of the final module mark

6th Semester

ADVANCED COMPUTER ARCHITECTURE

Aims

This course examines the techniques and underlying principles that are used to design high-performance computers and processors. Particular emphasis is placed on understanding the trade-offs involved when making design decisions at the architectural level. A range of processor architectures are explored and contrasted. In each case we examine their merits and limitations and how ultimately the ability to scale performance is restricted.

Lectures

Introduction. The impact of technology scaling and market trends.
Fundamentals of Computer Design. Amdahl's law, energy/performance trade-offs, ISA design.
Advanced pipelining. Pipeline hazards; exceptions; optimal pipeline depth; branch prediction; the branch target buffer [2 lectures]
Superscalar techniques. Instruction-Level Parallelism (ILP); superscalar processor architecture [2 lectures]
Software approaches to exploiting ILP. VLIW architectures; local and global instruction scheduling techniques; predicated instructions and support for speculative compiler optimisations.
Multithreaded processors. Coarse-grained, fine-grained, simultaneous multithreading
The memory hierarchy. Caches; programming for caches; prefetching [2 lectures]
Vector processors. Vector machines; short vector/SIMD instruction set extensions; stream processing
Chip multiprocessors. The communication model; memory consistency models; false sharing; multiprocessor memory hierarchies; cache coherence protocols; synchronization [2 lectures]
On-chip interconnection networks. Bus-based interconnects; on-chip packet switched networks
Special-purpose architectures. Converging approaches to computer design
Objectives
At the end of the course students should

understand what determines processor design goals;
appreciate what constrains the design process and how architectural trade-offs are made within these constraints;
be able to describe the architecture and operation of pipelined and superscalar processors, including techniques such as branch prediction, register renaming and out-of-order execution;
have an understanding of vector, multithreaded and multi-core processor architectures;
for the architectures discussed, understand what ultimately limits their performance and application domain.
Recommended reading
* Hennessy, J. and Patterson, D. (2012). Computer architecture: a quantitative approach. Elsevier (5th ed.) ISBN 9780123838728. (the 3rd and 4th editions are also good)

CRYPTOGRAPHY

Aims
This course provides an overview of basic modern cryptographic techniques and covers essential concepts that users of cryptographic standards need to understand to achieve their intended security goals.

Lectures
Cryptography. Overview, private vs. public-key ciphers, MACs vs. signatures, certificates, capabilities of adversary, Kerckhoffs' principle.
Classic ciphers. Attacks on substitution and transposition ciphers, Vigenére. Perfect secrecy: one-time pads.
Private-key encryption. Stream ciphers, pseudo-random generators, attacking linear-congruential RNGs and LFSRs. Semantic security definitions, oracle queries, advantage, computational security, concrete-security proofs.
Block ciphers. Pseudo-random functions and permutations. Birthday problem, random mappings. Feistel/Luby-Rackoff structure, DES, TDES, AES.
Chosen-plaintext attack security. Security with multiple encryptions, randomized encryption. Modes of operation: ECB, CBC, OFB, CNT.
Message authenticity. Malleability, MACs, existential unforgeability, CBC-MAC, ECBC-MAC, CMAC, birthday attacks, Carter-Wegman one-time MAC.
Authenticated encryption. Chosen-ciphertext attack security, ciphertext integrity, encrypt-and-authenticate, authenticate-then-encrypt, encrypt-then-authenticate, padding oracle example, GCM.
Secure hash functions. One-way functions, collision resistance, padding, Merkle-Damgård construction, sponge function, duplex construct, entropy pool, SHA standards.
Applications of secure hash functions. HMAC, stream authentication, Merkle tree, commitment protocols, block chains, Bitcoin.
Key distribution problem. Needham-Schroeder protocol, Kerberos, hardware-security modules, public-key encryption schemes, CPA and CCA security for asymmetric encryption.
Number theory, finite groups and fields. Modular arithmetic, Euclid's algorithm, inversion, groups, rings, fields, GF(2n), subgroup order, cyclic groups, Euler's theorem, Chinese remainder theorem, modular roots, quadratic residues, modular exponentiation, easy and difficult problems. [2 lectures]
Discrete logarithm problem. Baby-step-giant-step algorithm, computational and decision Diffie-Hellman problem, DH key exchange, ElGamal encryption, hybrid cryptography, Schnorr groups, elliptic-curve systems, key sizes. [2 lectures]
Trapdoor permutations. Security definition, turning one into a public-key encryption scheme, RSA, attacks on "textbook" RSA, RSA as a trapdoor permutation, optimal asymmetric encryption padding, common factor attacks.
Digital signatures. One-time signatures, RSA signatures, Schnorr identification scheme, ElGamal signatures, DSA, PS3 hack, certificates, PKI.
Objectives
By the end of the course students should

be familiar with commonly used standardized cryptographic building blocks;
be able to match application requirements with concrete security definitions and identify their absence in naive schemes;
understand various adversarial capabilities and basic attack algorithms and how they affect key sizes;
understand and compare the finite groups most commonly used with discrete-logarithm schemes;
understand the basic number theory underlying the most common public-key schemes, and some efficient implementation techniques.
Recommended reading
Katz, J., Lindell, Y. (2015). Introduction to modern cryptography. Chapman and Hall/CRC (2nd ed.).

E-COMMERCE

Aims
This course aims to give students an outline of the issues involved in setting up an e-commerce site.

Lectures
The history of electronic commerce. Mail order; EDI; web-based businesses, credit card processing, PKI, identity and other hot topics.
Network economics. Real and virtual networks, supply-side versus demand-side scale economies, Metcalfe's law, the dominant firm model, the differentiated pricing model Data Protection Act, Distance Selling regulations, business models.
Web site design. Stock and price control; domain names, common mistakes, dynamic pages, transition diagrams, content management systems, multiple targets.
Web site implementation. Merchant systems, system design and sizing, enterprise integration, payment mechanisms, CRM and help desks. Personalisation and internationalisation.
The law and electronic commerce. Contract and tort; copyright; binding actions; liabilities and remedies. Legislation: RIP; Data Protection; EU Directives on Distance Selling and Electronic Signatures.
Putting it into practice. Search engine interaction, driving and analysing traffic; dynamic pricing models. Integration with traditional media. Logs and audit, data mining modelling the user. collaborative filtering and affinity marketing brand value, building communities, typical behaviour.
Finance. How business plans are put together. Funding Internet ventures; the recent hysteria; maximising shareholder value. Future trends.
UK and International Internet Regulation. Data Protection Act and US Privacy laws; HIPAA, Sarbanes-Oxley, Security Breach Disclosure, RIP Act 2000, Electronic Communications Act 2000, Patriot Act, Privacy Directives, data retention; specific issues: deep linking, Inlining, brand misuse, phishing.
Objectives
At the end of the course students should know how to apply their computer science skills to the conduct of e-commerce with some understanding of the legal, security, commercial, economic, marketing and infrastructure issues involved.

Recommended reading
Shapiro, C. and Varian, H. (1998). Information rules. Harvard Business School Press.

Additional reading:

Standage, T. (1999). The Victorian Internet. Phoenix Press. Klemperer, P. (2004). Auctions: theory and practice. Princeton Paperback ISBN 0-691-11925-2.

MACHINE LEARNING AND BAYESIAN INFERENCE

Aims

The Part 1B course Artificial Intelligence introduced simple neural networks for supervised learning, and logic-based methods for knowledge representation and reasoning. This course has two aims. First, to provide a rigorous introduction to machine learning, moving beyond the supervised case and ultimately presenting state-of-the-art methods. Second, to provide an introduction to the wider area of probabilistic methods for representing and reasoning with knowledge.

Lectures

Introduction to learning and inference. Supervised, unsupervised, semi-supervised and reinforcement learning. Bayesian inference in general. What the naive Bayes method actually does. Review of backpropagation. Other kinds of learning and inference. [1 lecture]

How to classify optimally. Treating learning probabilistically. Bayesian decision theory and Bayes optimal classification. Likelihood functions and priors. Bayes theorem as applied to supervised learning. The maximum likelihood and maximum a posteriori hypotheses. What does this teach us about the backpropagation algorithm? [2 lectures]

Linear classifiers I. Supervised learning via error minimization. Iterative reweighted least squares. The maximum margin classifier. [2 lectures]

Gaussian processes. Learning and inference for regression using Gaussian process models. [2 lectures]

Support vector machines (SVMs). The kernel trick. Problem formulation. Constrained optimization and the dual problem. SVM algorithm. [2 lectures]

Practical issues. Hyperparameters. Measuring performance. Cross-validation. Experimental methods. [1 lecture]

Linear classifiers II. The Bayesian approach to neural networks. [1 lecture]

Unsupervised learning I. The k-means algorithm. Clustering as a maximum likelihood problem. [1 lecture]

Unsupervised learning II. The EM algorithm and its application to clustering. [1 lecture]

Bayesian networks I. Representing uncertain knowledge using Bayesian networks. Conditional independence. Exact inference in Bayesian networks. [2 lectures]

Bayesian networks II. Markov random fields. Approximate inference. Markov chain Monte Carlo methods. [1 lecture]

Objectives

At the end of this course students should:

Understand how learning and inference can be captured within a probabilistic framework, and know how probability theory can be applied in practice as a means of handling uncertainty in AI systems.

Understand several algorithms for machine learning and apply those methods in practice with proper regard for good experimental practice.

Recommended reading

If you are going to buy a single book for this course I recommend:

* Bishop, C.M. (2006). Pattern recognition and machine learning. Springer.

The course text for Artificial Intelligence I:

Russell, S. and Norvig, P. (2010). Artificial intelligence: a modern approach. Prentice Hall (3rd ed.).

covers some relevant material but often in insufficient detail. Similarly:

Mitchell, T.M. (1997). Machine Learning. McGraw-Hill.

gives a gentle introduction to some of the course material, but only an introduction.

Recently a few new books have appeared that cover a lot of relevant ground well. For example:

Barber, D. (2012). Bayesian Reasoning and Machine Learning. Cambridge University Press.
Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press.
Murphy, K.P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

OPTIMISING COMPILERS

Aims
The aims of this course are to introduce the principles of program optimisation and related issues in decompilation. The course will cover optimisations of programs at the abstract syntax, flowgraph and target-code level. It will also examine how related techniques can be used in the process of decompilation.

Lectures
Introduction and motivation. Outline of an optimising compiler. Optimisation partitioned: analysis shows a property holds which enables a transformation. The flow graph; representation of programming concepts including argument and result passing. The phase-order problem.
Kinds of optimisation. Local optimisation: peephole optimisation, instruction scheduling. Global optimisation: common sub-expressions, code motion. Interprocedural optimisation. The call graph.
Classical dataflow analysis. Graph algorithms, live and avail sets. Register allocation by register colouring. Common sub-expression elimination. Spilling to memory; treatment of CSE-introduced temporaries. Data flow anomalies. Static Single Assignment (SSA) form.
Higher-level optimisations. Abstract interpretation, Strictness analysis. Constraint-based analysis, Control flow analysis for lambda-calculus. Rule-based inference of program properties, Types and effect systems. Points-to and alias analysis.
Target-dependent optimisations. Instruction selection. Instruction scheduling and its phase-order problem.
Decompilation. Legal/ethical issues. Some basic ideas, control flow and type reconstruction.
Objectives
At the end of the course students should

be able to explain program analyses as dataflow equations on a flowgraph;
know various techniques for high-level optimisation of programs at the abstract syntax level;
understand how code may be re-scheduled to improve execution speed;
know the basic ideas of decompilation.
Recommended reading
* Nielson, F., Nielson, H.R. and Hankin, C.L. (1999). Principles of program analysis. Springer. Good on part A and part B.
Appel, A. (1997). Modern compiler implementation in Java/C/ML (3 editions).
Muchnick, S. (1997). Advanced compiler design and implementation. Morgan Kaufmann.
Wilhelm, R. (1995). Compiler design. Addison-Wesley.
Aho, A.V., Sethi, R. and Ullman, J.D. (2007). Compilers: principles, techniques and tools. Addison-Wesley (2nd ed.).

QUANTUM COMPUTING

Aims

The principal aim of the course is to introduce students to the basics of the quantum model of computation. The model will be used to study algorithms for searching, factorisation and quantum chemistry as well as other important topics in quantum information such as cryptography and super-dense coding. Issues in the complexity of computation will also be explored. A second aim of the course is to introduce student to near-term quantum computing. To this end, error-correction and adiabatic quantum computing are studied.

Lectures

Bits and qubits. Introduction to quantum states and measurements with motivating examples. Comparison with discrete classical states.
Linear algebra. Review of linear algebra: vector spaces, linear operators, Dirac notation, the tensor product.
The postulates of quantum mechanics. Postulates of quantum mechanics, incl. evolution and measurement.
Important concepts in quantum mechanics. Entanglement, distinguishing orthogonal and non-orthogonal quantum states, no-cloning and no signalling.
The quantum circuit model. The circuit model of quantum computation. Quantum gates and circuits. Universality of the quantum circuit model, and efficient simulation of arbitrary two-qubit gates with a standard universal set of gates.
Some applications of quantum information. Applications of quantum information (other than quantum computation): quantum key distribution, superdense coding and quantum teleportation.
Deutsch-Jozsa algorithm. Introducing Deutsch's problem and Deutsch's algorithm leading onto its generalisation, the Deutsch-Jozsa algorithm.
Quantum search. Grover's search algorithm: analysis and lower bounds.
Quantum Fourier Transform and Quantum Phase Estimation. Definition of the Quantum Fourier Transform (QFT), and efficient representation thereof as a quantum circuit. Application of the QFT to enable Quantum Phase Estimation (QPE).
Application 1 of QFT / QPE: Factoring. Shor's algorithm: reduction of factoring to period finding and then using the QFT for period finding.
Application 2 of QFT / QPE: Quantum Chemistry. Efficient simulation of quantum systems, and applications to real-world problems in quantum chemistry.
Quantum complexity. Quantum complexity classes and their relationship to classical complexity. Comparison with probabilistic computation.
Quantum error correction. Introducing the concept of quantum error correction required for the following lecture on fault-tolerance.
Fault tolerant quantum computing. Elements of fault tolerant computing; the threshold theorem for efficient suppression of errors.
Adiabatic quantum computing and quantum optimisation. The quantum adiabatic theorem, and adiabatic optimisation. Quantum annealing and D-Wave.
Case studies in near-term quantum computation. Examples of state-of-the-art quantum algorithms and computers, including superconducting and networked quantum computers.

Objectives
At the end of the course students should:

understand the quantum model of computation and the basic principles of quantum mechanics;
be familiar with basic quantum algorithms and their analysis;
be familiar with basic quantum protocols such as teleportation and superdense coding;
see how the quantum model relates to classical models of deterministic and probabilistic computation.
appreciate the importance of efficient error-suppression if quantum computation is to yield an advantage over classical computation.
gain a general understanding of the important topics in near-term quantum computing, including adiabatic quantum computing.
Recommended reading
Books:
Nielsen M.A., Chuang I.L. (2010). Quantum Computation and Quantum Information. Cambridge University Press.
Mermin N.D. (2007). Quantum Computer Science: An Introduction. Cambridge University Press.
McGeoch, C. (2014). Adiabatic Quantum Computation and Quantum Annealing Theory and Practice. Morgan and Claypool. https://ieeexplore.ieee.org/document/7055969

Papers:

Braunstein S.L. (2003). Quantum computation tutorial. Available at:
https://www-users.cs.york.ac.uk/~schmuel/comp/comp_best.pdf
Aharonov D., Quantum computation [arXiv:quant-ph/9812037]
Albash T., Adiabatic Quantum Computing https://arxiv.org/pdf/1611.04471.pdf
McCardle S. et al, Quantum computational chemistry https://arxiv.org/abs/1808.10402

Other lecture notes:

Andrew Childs (University of Maryland): http://cs.umd.edu/~amchilds/qa/

RANDOMISED ALGORITHMS

Aims:
The aim of this course is to introduce advanced techniques in the design and analysis algorithms, with a strong focus on randomised algorithms. It covers essential tools and ideas from probability, optimisation and graph theory, and develops this knowledge through a variety of examples of algorithms and processes. This course will demonstrate that randomness is not only an important design technique which often leads to simpler and more elegant algorithms, but may also be essential in settings where time and space are restricted.

Lectures:
Introduction. Course Overview. Why Randomised Algorithms? A first Randomised Algorithm for the MAX-CUT problem. [1 Lecture]

Concentration Inequalities. Moment-Generating Functions and Chernoff Bounds. Extension: Method of Bounded Independent Differences. Applications: Balls-into-Bins, Quick-Sort and Load Balancing. [approx. 2 Lectures]

Markov Chains and Mixing Times. Random Walks on Graphs. Application: Randomised Algorithm for the 2-SAT problem. Mixing Times of Markov Chains. Application: Load Balancing on Networks. [approx. 2 Lectures]

Linear Programming and Applications. Definitions and Applications. Formulating Linear Programs. The Simplex Algorithm. Finding Initial Solutions. How to use Linear Programs and Branch & Bound to Solve a Classical TSP instance. [approx. 3 Lectures]

Randomised Approximation Algorithms. Randomised Approximation Schemes. Linearity of Expectations, Derandomisation. Deterministic and Randomised Rounding of Linear Programs. Applications: MAX3-SAT problem, Weighted Vertex Cover, Weighted Set Cover, MAX-SAT. [approx. 2 Lectures]

Spectral Graph Theory and Clustering. Eigenvalues of Graphs and Matrices: Relations between Eigenvalues and Graph Properties, Spectral Graph Drawing. Spectral Clustering: Conductance, Cheeger's Inequality. Spectral Partitioning Algorithm [approx. 2 Lectures]

Objectives:
By the end of the course students should be able to:

learn how to use randomness in the design of algorithms, in particular, approximation algorithms;
learn the basics of linear programming, integer programming and randomised rounding;
apply randomisation to various problems coming from optimisation, machine learning and data science;
use results from probability theory to analyse the performance of randomised algorithms.

Recommended reading
* Michael Mitzenmacher and Eli Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis., Cambridge University Press, 2nd edition.

* David P. Williamson and David B. Shmoys. The Design of Approximation Algorithms, Cambridge University Press, 2011

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. and Stein, C. (2009). Introduction to Algorithms. MIT Press (3rd ed.). ISBN 978-0-262-53305-8

CLOUD COMPUTING

Aims

This module aims to teach students the fundamentals of Cloud Computing covering topics such as virtualization, data centres, cloud resource management, cloud storage and popular cloud applications including batch and data stream processing. Emphasis is given on the different backend technologies to build and run efficient clouds and the way clouds are used by applications to realise computing on demand. The course will include practical tutorials on cloud infrastructure technologies. Students will be assessed via a Cloud-based coursework project.

Lectures

Introduction to Cloud Computing
Data centres
Virtualization I
Virtualization II
MapReduce
MapReduce advanced
Resource management for virtualized data centres
Cloud storage
Cloud-based data stream processing

Objectives

By the end of the course students should:

understand how modern clouds operate and provide computing on demand;
understand about cloud availability, performance, scalability and cost;
know about cloud infrastructure technologies including virtualization, data centres, resource management and storage;
know how popular applications such as batch and data stream processing run efficiently on clouds;

Assessment

Assignment 1, worth 55% of the final mark. Develop batch processing applications running on a cluster assessed through automatic testing, code inspection and a report.
Assignment 2, worth 30% of the final mark: Design and develop a framework for running the batch processing applications from Assignment 1 on a cluster according to certain resource allocation policies. This assigment will be assessed by an expert, testing, code inspection and a report.
Assignment 3, worth 15% of the final mark: Write two individual project reports describing your work for assignments 1 and 2. Write two sets of scripts to manage and run your code. Reports and scripts will be submitted with each assignment.

Recommended reading

Marinescu, D.C. Cloud Computing, Theory and Practice. Morgan Kaufmann.
Barham, P., et. al. (2003). “Xen and the Art of Virtualization”. In Proceedings of SOSP 2003.
Charkasova, L., Gupta, D. and Vahdat, A. (2007). “Comparison of the Three CPU Schedulers in Xen”. In SIGMETRICS 2007.

Dean, J. and Ghemawat, S. (2004). “MapReduce: Simplified Data Processing on Large Clusters”. In Proceedings of OSDI 2004.
Zaharia, M, et al. (2008). “Improving MapReduce Performance in Heterogeneous Environments”. In Proceedings of OSDI 2008.
Hindman, A., et al. (2011). “Mesos: A Platform for Fine-Grained Resource Sharing in Data Center”. In Proceedings of NSDI 2011.
Schwarzkopf, M., et al. (2013). “Omega: Flexible, Scalable Schedulers for Large Compute Clusters”. In EuroSys 2013.
Ghemawat, S. (2003). “The Google File System”. In Proceedings of SOSP 2003.
Chang, F. (2006). “Bigtable: A Distributed Storage System for Structured Data”. In Proceedings of OSDI 2006.
Fernandez, R.C., et al. (2013). “Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management”. In SIGMOD 2013.

COMPUTER SYSTEMS MODELLING

Aims

The aims of this course are to introduce the concepts and principles of mathematical modelling

and simulation, with particular emphasis on using queuing theory and control theory for understanding the behaviour of computer and communications systems.

Syllabus

1. Overview of computer systems modeling using both analytic techniques and simulation.

2. Introduction to discrete event simulation.

a. Simulation techniques

b. Random number generation methods

c. Statistical aspects: confidence intervals, stopping criteria

d. Variance reduction techniques.

3. Stochastic processes (builds on starred material in Foundations of Data Science)

a. Discrete and continuous stochastic processes.

b. Markov processes and Chapman-Kolmogorov equations.

c. Discrete time Markov chains.

d. Ergodicity and the stationary distribution.

e. Continuous time Markov chains.

f. Birth-death processes, flow balance equations.

g. The Poisson process.

4. Queueing theory

a. The M/M/1 queue in detail.

b. The equilibrium distribution with conditions for existence and common performance metrics.

c. Extensions of the M/M/1 queue: the M/M/k queue, the M/M/infinity queue.

d. Queueing networks. Jacksonian networks.

e. The M/G/1 queue.

5. Signals, systems, and transforms

a. Discrete- and continuous-time convolution.

b. Signals. The complex exponential signal.

c. Linear Time-Invariant Systems. Modeling practical systems as an LTI system.

d. Fourier and Laplace transforms.

6. Control theory.

a. Controlled systems. Modeling controlled systems.

b. State variables. The transfer function model.

c. First-order and second-order systems.

d. Feedback control. PID control.

e. Stability. BIBO stability. Lyapunov stability.

f. Introduction to Model Predictive Control.

Objectives

At the end of the course students should

· Be aware of different approaches to modeling a computer system; their pros and cons

· Be aware of the issues in building a simulation of a computer system and analysing the results obtained

· Understand the concept of a stochastic process and how they arise in practice

- Be able to build simple Markov models and understand the critical modelling assumptions
- Be able to solve simple birth-death processes
- Understand and use M/M/1 queues to model computer systems
- Be able to model a computer system as a linear time-invariant system
- Understand the dynamics of a second-order controlled system
- Design a PID control for an LTI system
- Understand what is meant by BIBO and Lyapunov stability

Assessment

There will be one programming assignment to build a discrete event simulator and using it to simulate an M/M/1 and an M/D/1 queue (worth 15% of the final marks). There will also be two one-hour in-person assessments for the course on: Stochastic processes and Queueing theory (40%) and Signals, Systems, Transforms, and Control Theory (45%).

Reference books

Keshav, S. (2012)*. Mathematical Foundations of Computer Networking. Addison-Wesley. A customized PDF will be made available to class participants.

Kleinrock, L. (1975). Queueing systems, vol. 1. Theory. Wiley.

Kraniauskas, Peter. Transforms in signals and systems. Addison-Wesley Longman, 1992.

Jain, R. (1991). The art of computer systems performance analysis. Wiley.

COMPUTING EDUCATION

Aims

This module considers various aspects of the teaching and learning of computer science in school, including practical experience, and provides an introduction to the field of computing education research. It is offered by the Raspberry Pi Computing Education Research Centre, part of the Department of Computer Science and Technology, in conjunction with local schools in the Cambridge area. Students will have the opportunity to observe, plan and deliver a lesson, and explore the teaching of computing in a secondary education setting. The lesson taught will be on a topic within the computing/computer science curriculum in England but will be negotiated with the teacher mentoring the school placement. In the sessions in the Department, students will be introduced to elements of pedagogy and assessment alongside current issues in computing education. The course can also serve as a useful introduction to research in computing education for those interested in this area. Assessment will include lesson planning, presentations, an in-person viva and a 3000-word report which will include evidence of engagement with relevant research underpinning the lessons and activities undertaken. An information session relating to the module will be available in the preceding Easter Term. The arrangement of school placements and DBS checks will be carried out in Michaelmas Term, so students should be prepared to engage in these preparatory activities.

Lectures

This is not a traditional lecture course and the term will be structured as follows:

Week 1: Introduction to computing education & visit to school placement
Week 2: In school (observation, supporting teacher)
Week 3: In school (observation, supporting teacher)
Week 4: In school (co-teach / small group teaching)
Week 5: Computing pedagogy and assessment (school half-term)
Week 6: In school (co-teach / small group teaching)
Week 7: In school (teach part/whole lesson)
Week 8: Presentations

Objectives

By the end of the course students should:

gain experience of computing education in practice
demonstrate an ability to communicate an aspect of computer science clearly and effectively
be able to plan and design a teaching activity/lesson with consideration of the audience
be able to thoroughly evaluate the design and implementation of a teaching/activity lesson
be able to demonstrate an understanding of relevant theories of learning and pedagogy from the literature and how they apply to the learning/teaching of the topic under consideration
gain an understanding of the breadth and depth of the field of computing education research

Class Size

This module can accommodate up to 10 Part II students.

Recommended reading
Brown, N. C., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education (TOCE), 14(2), 1-22. https://doi.org/10.1145/2602484

Sentance, S., Barendsen, E., Howard, N. R., & Schulte, C. (Eds.). (2023). Computer science education: Perspectives on teaching and learning in school. Bloomsbury Publishing.

Grover, S. (Ed). 2020. Computer Science in K-12: An A-to-Z Handbook on Teaching Programming. Edfinity. https://www.shuchigrover.com/atozk12cs/

Raspberry Pi Foundation (2022). Hello World Big Book of Pedagogy. https://www.raspberrypi.org/hello-world/issues/the-big-book-of-computing-pedagogy

Assessment
Week 2: Reflections on an observed lesson (10%, graded out of 20)
Week 4: Submission of a lesson plan outline (10% graded out of 20)
Week 8: Delivery of an oral presentation relating to the lesson delivery (10% graded out of 20)
After course completion: 3000 word report (60%), graded out of 20
Viva with assessor (10%) (pass/fail)
Further Information
We will find placements for students in secondary schools within cycling or bus distance from the Department/centre of Cambridge. There will be a timetabled slot for the course that students can use to go to their school placement, but students may need to be flexible and liaise with the school to find the best time for working with a specific class.

DEEP NEURAL NETWORKS

Objectives
You will gain detailed knowledge of

Current understanding of generalization in neural networks vs classical statistical models.
Optimization procedures for neural network models such as stochastic gradient descent and ADAM.
Automatic differentiation and at least one software framework (PyTorch, TensorFolow) as well as an overview of other software approaches.
Architectures that are deployed to deal with different data types such as images or sequences including a. convolutional networks b. recurrent networks and c. transformers.
In addition you will gain knowledge of more advanced topics reflecting recent research in machine learning. These change from year to year, examples include:

Approaches to unsupervised learning including autoencoders and self-supervised learning..
Techniques for deploying models in low data regimes such as transfer learning and meta-learning.
Techniques for propagating uncertainty such as Bayesian neural networks.
Reinforcement learning and its applications to robotics or games.
Graph Neural Networks and Geometric Deep Learning.

Teaching Style
The start of the course will focus on the latest undertanding of current theory of neural networks, contrasting with previous classical understandings of generalization performance. Then we will move to practical examples of network architectures and deployment. We will end with more advanced topics reflecting current research, mostly delivered by guest lecturers from the Department and beyond.

Schedule
Week 1
Two lectures: Generalization and Approximation with Neural Networks

Week 2
Two lectures: Optimization: Stochastic Gradient Descent and ADAM

Week 3
Two lectures: Hardware Acceleration and Convolutional Neural Networks

Week 4
Two lectures: Vanishing Gradients, Recurrent and Residual Networks, Sequence Modeling.

Week 5
Two lectures: Attention, The Transformer Architecture, Large Language Models

Week 6-8
Lectures and guest lectures from the special topics below.

Special topics
Self-Supervised Learning, AutoEncoders, Generative Modeling of Images
Hardware Implementations
Reinforcement learning
Transfer learning and meta-learning
Uncertainty and Bayesian Neural Networks
Graph Neural Networks
Assessment
Assessment involves solving a number of exercises in a jupyter notebook environment. Familiarity with the python programming language is assumed. The exercises rely on the pytorch deep learning framework, the syntax of which we don't cover in detail in the lectures, students are referred to documentation and tutorials to learn this component of the assessment.

Assignment 1

30% of the total marks, with guided exercises.

Assignment 2

70% of the total marks, a combination of guided exercises and a more exploratory mini-project.

EXTENDED REALITY

Syllabus & Schedule
Week 1
Introduction
Course schedule and concept
The history of AR/VR/XR
Applications of AR/VR/XR
Overview of the XR pipeline – 3 main components of XR
Display technologies and rendering
Machine perception and input interfaces
Content generation – geometry, appearance, motion
Overview of the AR/VR/XR frameworks
Practical 1
Project structure
Project themes
Introduction to the XR programming platform
Week 2
3D scene representations and rendering
Object representations with polygonal meshes
Efficient rendering via rasterization
Practical XR rendering of textured geometries
Tracking and pose estimation for XR
Rigid transforms
Camera pose estimation
Body/ hand tracking
Practical 2
Project draft proposal due
Project feedback
Rendering and displaying a first object
Week 3
3D scene capture and understanding
Depth estimation
3D geometry capture
Appearance acquisition
Lighting estimation and relighting
Lighting models for 3D scenes
Estimating lighting in a scene
Relighting rendered models
Practical 3
Acquiring and utilizing depth from sensors
Relighting a displayed object
Week 4
Physically-based simulation of rigid objects

Rigid-body dynamics
Collision detection
Time integration
Physically-based simulation of non-rigid volumetric phenomena
Particle systems
Basic fluid dynamics
Practical 4
Practical due
Project prototype due
Project check-in
Simulating and rendering a simple object
Week 5
Advanced topics in XR I: AR/VR displays technologies
Micro LED-based displays
Waveguide-based displays
Next generation displays
Week 6
Advanced topics in XR II – Mobile AR
Computer vision on limited hardware
UX/UI challenges
Content challenges
Project due

Assessment
A practical exercise, worth 20% of the mark. This will cover the basics of AR/VR development and some of the practical techniques the students will use in their projects. This is individual work. No mobile device or GPU hours are needed.
An AR/VR project worth 80% of the marks:
Course projects will be designed by the students following the possible project themes proposed by the lecturer and will be checked by the lecturer for appropriateness.
The students will form groups of 2-3 to design, implement, report, and present the course project.
The final mark will be an implementation mark (40%), a report mark (20%), and a presentation/demo mark (20%). Each team member will be evaluated based on their contribution.
Each project will have extensions to be completed only by the ACS students. Each student will write a different part of the report, whose author will be clearly marked. Each student will further summarise her/his contributions to the project in the same report.
Note: Submission is by one submission per group but work by each individual must be clearly labelled.

FEDERATED LEARNING: THEORY AND PRACTICE

Objectives
This course aims to extend the machine learning knowledge available to students in Part I (or present in typical undergraduate degrees in other universities), and allow them to understand how these concepts can manifest in a decentralized setting. The course will consider both theoretical (e.g., decentralized optimization) and practical (e.g., networking efficiency) aspects that combine to define this growing area of machine learning.

At the end of the course students should:

Understand popular methods used in federated learning
Be able to construct and scale a simple federated system
Have gained an appreciation of the core limitations to existing methods, and the approaches available to cope with these issues
Developed an intuition for related technologies like differential privacy and secure aggregation, and are able to use them within typical federated settings
Can reason about the privacy and security issues with federated systems
Lectures
Course Overview. Introduction to Federated Learning.
Decentralized Optimization.
Statistical and Systems Heterogeneity.
Variations of Federated Aggregation.
Secure Aggregation.
Differential Privacy within Federated Systems.
Extensions to Federated Analytics.
Applications to Speech, Video, Images and Robotics.
Lab sessions
Federating a Centralized ML Classifier.
Behaviour under Heterogeneity.
Scaling a Federated Implementation.
Exploring Privacy with Federated Settings
Recommended Reading
Readings will be assigned for each lecture. Readings will be taken from either research papers, tutorials, source code or blogs that provide more comprehensive treatment of taught concepts.

Assessment - Part II Students
Four labs are performed during the course, and students receive 10% of their total grade for work done as part of each lab. (For a total of 40% of the total grade from lab work alone). Labs will primarily provide hands-on teaching opportunities, that are then utilized within the lab assignment which is completed outside of the lab contact time. MPhil and Part III students will be given additional questions to answer within their version of the lab assignment which will differ from the assignment given to Part II CST students.

The remainder of the course grade (60%) will be given based on a hands-on project that applies the concepts taught in lectures and labs. This hands-on project will be assessed based on upon a combination of source code, related documentation and brief 8-minute pre-recorded talk that summarizes key project elements (any slides used are also submitted as part of the project). Please note, that in the case of Part II CST students, the talk itself is not examinable -- as such will be made optional to those students.

A range of possible practical projects will be described and offered to students to select from, or alternatively students may propose their own. MPhil and Part III students will select from a project pool that is separate from those offered to Part II CST students. MPhil and Part III projects will contain a greater emphasis on a research element, and the pre-recorded talks provided by this student group will focus on this research contribution. The project will be assessed on the level of student understanding demonstrated, the degree of difficulty, correctness of implementation -- and for Part III/MPhil students the additional criteria of the quality and execution of the research methodology, and depth and quality of results analysis.

This project can be done individually or in groups -- although individual projects will be strongly encouraged. It will be required the project is performed using a code repository that also will contain all documentation -- access to this repository will be shared with course staff (e.g., lecturer and TAs). Where needed, marks assigned to students within a group will be differentiated using this repository as an input. Furthermore if groups are formed, members must be either entirely from Part III/MPhil students or Part II CST, i.e., these two student groups should not mix to form a project group.

Projects will be made available publicly. A maximum word count for written contributions for the project will be enforced.

MOBILE HEALTH

Aims

The course aims to explore how wearable and mobile systems sensors can be used to gather data relevant to understand health, how the data can be analysed with advanced signal processing and machine learning and the performance of these systems in terms of diagnostics and disease progression detection.

Syllabus

Course Overview. Introduction to Mobile Health. Evaluation metrics and methodology. Basics of Signal Processing.
Inertial Measurement Units, Human Activity Recognition (HAR) and Gait Analysis and Machine Learning for IMU data.
Radios, Bluetooth, GPS and Cellular. Epidemiology and contact tracing, Social interaction sensing and applications. Location tracking in health monitoring and public health.
Audio Signal Processing. Voice and Speech Analysis: concepts and data analysis. Body Sounds analysis.
Photoplethysmogram and Light sensing for health (heart and sleep)
Contactless and wireless behaviour and physiological monitoring
Generative Models for Wearable Health Data
Topical Guest Lectures

Objectives

The course aims to explore how wearable and mobile systems sensors can be used to gather data relevant to understand health, how the data can be analysed with advanced signal processing and machine learning and the performance of these systems in terms of diagnostics and disease progression detection.

Roughly, each lecture contains a theory part about the working of “sensor signals” or “data analysis methods” and an application part which contextualises the concepts.

At the end of the course students should: Understand how mobile/wearable sensors capture data and their working. Understand different approaches to acquiring and analysing sensor data from different types of sensors. Understand the concept of signal processing applied to time series data and their practical application in health. Be able to extract sensor data and analyse it with basic signal processing and machine learning techniques. Be aware of the different health applications of the various sensor techniques. The course will also touch on privacy and ethics implications of the approaches developed in an orthogonal fashion.

Recommended Reading

Please see Course Materials for recommended reading for each session.

Assessment - Part II students

Two assignments will be based on two datasets which will be provided to the students:

Assignment 1 (shared for Part II and Part III/MPhil): this will be based on a dataset and will be worth 40% of the final mark. The task of the assessment will be to perform pre-processing and basic data analysis in a "colab" and an answer sheet of no more than 1000 words.

Assignment 2 (Part II): This assignment (worth 60% of the final mark) will be a fuller analysis of a dataset focusing on machine learning algorithms and metrics. Discussion and interpretation of the findings will be reported in a colab and a report of no more than 1200 words.

MULTICORE SEMANTICS AND PROGRAMMING

Aims
In recent years multiprocessors have become ubiquitous, but building reliable concurrent systems with good performance remains very challenging. The aim of this module is to introduce some of the theory and the practice of concurrent programming, from hardware memory models and the design of high-level programming languages to the correctness and performance properties of concurrent algorithms.

Lectures
Part 1: Introduction and relaxed-memory concurrency [Professor P. Sewell]

Introduction. Sequential consistency, atomicity, basic concurrent problems. [1 block]
Concurrency on real multiprocessors: the relaxed memory model(s) for x86, ARM, and IBM Power, and theoretical tools for reasoning about x86-TSO programs. [2 blocks]
High-level languages. An introduction to C/C++11 and Java shared-memory concurrency. [1 block]
Part 2: Concurrent algorithms [Dr T. Harris]

Concurrent programming. Simple algorithms (readers/writers, stacks, queues) and correctness criteria (linearisability and progress properties). Advanced synchronisation patterns (e.g. some of the following: optimistic and lazy list algorithms, hash tables, double-checked locking, RCU, hazard pointers), with discussion of performance and on the interaction between algorithm design and the underlying relaxed memory models. [3 blocks]
Research topics, likely to include one hour on transactional memory and one guest lecture. [1 block]
Objectives
By the end of the course students should:

have a good understanding of the semantics of concurrent programs, both at the multprocessor level and the C/Java programming language level;
have a good understanding of some key concurrent algorithms, with practical experience.
Assessment - Part II Students
Two assignments each worth 50%

Recommended reading
Herlihy, M. and Shavit, N. (2008). The art of multiprocessor programming. Morgan Kaufmann.

BUSINESS STUDIES SEMINARS

Aims

This course is a series of seminars by former members and friends of the Laboratory about their real-world experiences of starting and running high technology companies. It is a follow on to the Business Studies course in the Michaelmas Term. It provides practical examples and case studies, and the opportunity to network with and learn from actual entrepreneurs.

Lectures

Eight lectures by eight different entrepreneurs.

Objectives

At the end of the course students should have a better knowledge of the pleasures and pitfalls of starting a high tech company.

Recommended reading

Lang, J. (2001). The high-tech entrepreneur's handbook: how to start and run a high-tech company. FT.COM/Prentice Hall.
Maurya, A. (2012). Running Lean: Iterate from Plan A to a Plan That Works. O'Reilly.
Osterwalder, A. and Pigneur, Y. (2010). Business Model Generation: A Handbook for Visionaires, Game Changers, and Challengers. Wiley.
Kim, W. and Mauborgne, R. (2005). Blue Ocean Strategy. Harvard Business School Press.

See also the additional reading list on the Business Studies web page.

HOARE LOGIC AND MODEL CHECKING

Aims
The course introduces two verification methods, Hoare Logic and Temporal Logic, and uses them to formally specify and verify imperative programs and systems.

The first aim is to introduce Hoare logic for a simple imperative language and then to show how it can be used to formally specify programs (along with discussion of soundness and completeness), and also how to use it in a mechanised program verifier.

The second aim is to introduce model checking: to show how temporal models can be used to represent systems, how temporal logic can describe the behaviour of systems, and finally to introduce model-checking algorithms to determine whether properties hold, and to find counter-examples.

Current research trends also will be outlined.

Lectures
Part 1: Hoare logic. Formal versus informal methods. Specification using preconditions and postconditions.
Axioms and rules of inference. Hoare logic for a simple language with assignments, sequences, conditionals and while-loops. Syntax-directedness.
Loops and invariants. Various examples illustrating loop invariants and how they can be found.
Partial and total correctness. Hoare logic for proving termination. Variants.
Semantics and metatheory Mathematical interpretation of Hoare logic. Semantics and soundness of Hoare logic.
Separation logic Separation logic as a resource-aware reinterpretation of Hoare logic to deal with aliasing in programs with pointers.
Part 2: Model checking. Models. Representation of state spaces. Reachable states.
Temporal logic. Linear and branching time. Temporal operators. Path quantifiers. CTL, LTL, and CTL*.
Model checking. Simple algorithms for verifying that temporal properties hold.
Applications and more recent developments Simple software and hardware examples. CEGAR (counter-example guided abstraction refinement).
Objectives
At the end of the course students should

be able to prove simple programs correct by hand and implement a simple program verifier;
be familiar with the theory and use of separation logic;
be able to write simple models and specify them using temporal logic;
be familiar with the core ideas of model checking, and be able to implement a simple model checker.
Recommended reading

Huth, M. and Ryan M. (2004). Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press (2nd ed.).

7th Semester

ADVANCED TOPICS IN COMPUTER ARCHITECTURE

Aims

This course aims to provide students with an introduction to a range of advanced topics in computer architecture. It will explore the current and future challenges facing the architects of modern computers. These will also be used to illustrate the many different influences and trade-offs involved in computer architecture.

Objectives

On completion of this module students should:

understand the challenges of designing and verifying modern microprocessors
be familiar with recent research themes and emerging challenges
appreciate the complex trade-offs at the heart of computer architecture

Syllabus

Each seminar will focus on a different topic. The proposed topics are listed below but there may be some minor changes to this:

Trends in computer architecture
State-of-the-art microprocessor design
Memory system design
Hardware reliability
Specification, verification and test (may be be replaced with a different topic)
Hardware security (2)
HW accelerators and accelerators for machine learning

Each two hour seminar will include three student presentations (15mins) questions (5mins) and a broader discussion of the topics (around 30mins). The last part of the seminar will include a short scene setting lecture (around 20mins) to introduce the following week's topic.

Assessment

Each week students will compare and contrast two of the main papers and submit a written summary and review in advance of each seminar (except when presenting).

Students will be expected to give a number of 15 minute presentations.

Essays and presentations will be marked out of 10. After dropping the lowest mark, the remaining marks will be scaled to give a final score out of 100.

Students will give at least one presentation during the course. They will not be required to submit an essay during the weeks they are presenting.

Each presentation will focus on a single paper from the reading list. Marks will be awarded for clarity and the communication of the paper's key ideas, an analysis of the work's strengths and weaknesses and the work’s relationship to related work and broader trends and constraints.

Recommended prerequisite reading

Patterson, D. A., Hennessy, J. L. (2017). Computer organization and design: The Hardware/software interface RISC-V edition Morgan Kaufmann. ISBN 978-0-12-812275-4.

Hennessy, J. and Patterson, D. (2012). Computer architecture: a quantitative approach. Elsevier (5th ed.) ISBN 9780123838728. (the 3rd and 4th editions are also relevant)

ADVANCED TOPICS IN PROGRAMMING LANGUAGES

Aims

This module explores various topics in programming languages beyond the scope of undergraduate courses. It aims to introduce students to ideas, results and techniques found in the literature and prepare them for research in the field.

Syllabus and structure

The module consists of eight two-hour seminars, each on a particular topic. Topics will vary from year to year, but may include, for example,

Abstract interpretation
Verified software
Metaprogramming
Behavioural types
Program synthesis
Verified compilation
Partial evaluation
Garbage collection
Dependent types
Automatic differentiation
Delimited continuations
Module systems

There will be three papers assigned for each topic, which students are expected to read before the seminar.

Each seminar will include three 20 minute student presentations (15 minutes + 5 minutes questions), time for general discussion of the topic, and a brief overview lecture for the following week's topic.

Before each seminar, except in weeks in which they give presentations, students will submit a short essay about two of the papers.

Objectives

On completion of this module, students should

be able to identify some major themes in programming language research
be familiar with some classic papers and recent advances
have an understanding of techniques used in the field

Assessment

Assessment consists of:

Presentation of one of the papers from the reading list (typically once or twice in total for each student, depending on class numbers)

One essay per week (except on the first week and on presentation weeks)

All essays and presentations carry equal numbers of marks.

Essay marks are awarded for understanding, for insight and analysis, and for writing quality. Essays should be around 1500 words (with a lower limit of 1450 and upper limit of 1650). Presentation marks are awarded for clarity, for effective communication, and for selection and organisation of topics.

There will be seven submissions (essays or presentations) in total and, as in other courses, the lowest mark for each student will be disregarded when computing the final mark.

Marking, deadlines and extensions will be handled in accordance with the MPhil Assessment Guidelines.

Recommended reading material and resources
Research and survey papers from programming language conferences and journals (e.g. POPL, PLDI, TOPLAS, FTPL) will be assigned each week. General background material may be found in:

- Types and Programming Languages (Benjamin C. Pierce)
  The MIT Press
  ISBN 0-262-16209-1

- Advanced Topics in Types and Programming Languages (ed. Benjamin C. Pierce)
  The MIT Press
  ISBN 0-262-16228-8

- Practical Foundations for Programming Languages (Robert Harper)
  Cambridge University Press
  9781107150300

AFFECTIVE ARTIFICIAL INTELLIGENCE

Synopsis
Affective Artificial Intelligence (Affective AI) aims to imbue machines with social and emotional intelligence (EQ). More specifically, Affective AI aims to create artificially intelligent systems and machines that can recognize, interpret, process, and simulate human social signals and behaviours, expressions, and emotions, to enhance human-AI interaction and communication. \r\n\r\nTo achieve this goal, Affective AI draws upon various scientific disciplines, including machine learning, computer vision, speech / natural language / signal processing, psychology and cognitive science, and ethics and social sciences.

Background & Aims
Affective Artificial Intelligence (Affective AI) aims to imbue machines with social and emotional intelligence (EQ). More specifically, Affective AI aims to create artificially intelligent systems and machines that can recognize, interpret, process, and simulate human social signals and behaviours, expressions, and emotions, to enhance human-AI interaction and communication.

To achieve this goal, Affective AI draws upon various scientific disciplines, including machine learning, computer vision, speech / natural language / signal processing, psychology and cognitive science, and ethics and social sciences.

Affective AI has direct applications in and implications for the design of innovative interactive technology (e.g., interaction with chat bots, virtual agents, robots), single and multi-user smart environments (e.g., in-car/ virtual / augmented / mixed reality, serious games), public speaking and cognitive training, and clinical and biomedical studies (e.g., autism, depression, pain).

The aim of this module is to impart knowledge and ability needed to make informed choices of models, data, and machine learning techniques for sensing, recognition, and generation of affective and social behaviour (e.g., smile, frown, head nodding/shaking, agreement/disagreement) in order to create Affectively intelligent AI systems, with a consideration for various ethical issues (e.g., privacy, bias) arising from the real-world deployment of these systems.

Syllabus
The following list provides a representative list of topics:

Introduction, definitions, and overview
Theories from various disciplines
Sensing from multiple modalities (e.g., vision, audio, bio signals, text)
Data acquisition and annotation
Signal processing / feature extraction

Learning / prediction / recognition and evaluation
Behaviour synthesis / generation (e.g., for embodied agents / robots)
Advanced topics and ethical considerations (e.g., bias and fairness)
Cross-disciplinary applications (via seminar presentations and discussions)
Guest lectures (diverse topics – changes each year)
Hands-on research and programming work (i.e., mini project & report)

Objectives
On completion of this module, students will:

understand and demonstrate knowledge in key characteristics of affectively intelligent AI, which include:
Recognition: How to equip Affective AI systems with capabilities of analysing facial expressions, vocal intonations, gestures, and other physiological signals to infer human affective states?
Generation: How to enable affectively intelligent AI simulate expressions of emotions in machines, allowing them to respond in a more human-like manner, such as virtual agents and humanoid robots, showing empathy or sympathy?
Adaptation and Personalization: How to enable affectively intelligent AI adapt system responses based on users' affective states and/or needs, or tailor interactions and experiences to individual users based on their expressivity / emotional profiles, personalities, and past interactions?
Empathetic Communication: How to design Affective AI systems to communicate with users in a way that demonstrates empathy, understanding, and sensitivity to their affective states and needs?
Ethical and societal considerations: What are the various human differences, ethical guidelines, and societal impacts that need to be considered when designing and deploying Affective AI to ensure that these systems respect users' privacy, autonomy, and well-being?
comprehend and apply (appropriate) methods for collection, analysis, representation, and evaluation of human affective and communicative behavioural data.
enhance programming skills for creating and implementing (components of) Affective AI systems.
demonstrate critical thinking, analysis and synthesis while deciding on 'when' and 'how' to incorporate human affect and social signals in a specific AI system context and gain practical experience in proposing and justifying computational solution(s) of suitable nature and scope.

Assessment - Part II Students
Seminar presentation: 20%
Participating in Q&A and discussions: 10%
Mid-term mini project report and presentation (as a team of 2): 10%
Final mini project report & code (as a team of 2): 60%

Assessment - MPhil / Part III Students
Seminar presentation: 20%
Participating in Q&A and discussions: 10%
Mid-term mini project report and presentation: 10%
Final mini project report & code: 60%
Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with ACS. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. More information will follow at the first lecture.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

CATEGORY THEORY

Aims
Category theory provides a unified treatment of mathematical properties and constructions that can be expressed in terms of 'morphisms' between structures. It gives a precise framework for comparing one branch of mathematics (organized as a category) with another and for the transfer of problems in one area to another. Since its origins in the 1940s motivated by connections between algebra and geometry, category theory has been applied to diverse fields, including computer science, logic and linguistics. This course introduces the basic notions of category theory: adjunction, natural transformation, functor and category. We will use category theory to organize and develop the kinds of structure that arise in models and semantics for logics and programming languages.

Syllabus
Introduction; some history. Definition of category. The category of sets and functions. Commutative diagrams. Examples of categories: preorders and monotone functions; monoids and monoid homomorphisms; a preorder as a category; a monoid as a category. Definition of isomorphism. Informal notion of a 'category-theoretic' property.
Terminal objects. The opposite of a category and the duality principle. Initial objects. Free monoids as initial objects.
Binary products and coproducts. Cartesian categories.
Exponential objects: in the category of sets and in general. Cartesian closed categories: definition and examples.
Intuitionistic Propositional Logic (IPL) in Natural Deduction style. Semantics of IPL in a cartesian closed preorder.
Simply Typed Lambda Calculus (STLC). The typing relation. Semantics of STLC types and terms in a cartesian closed category (ccc). The internal language of a ccc. The Curry-Howard-Lawvere correspondence.
Functors. Contravariance. Identity and composition for functors.
Size: small categories and locally small categories. The category of small categories. Finite products of categories.
Natural transformations. Functor categories. The category of small categories is cartesian closed.
Hom functors. Natural isomorphisms. Adjunctions. Examples of adjoint functors. Theorem characterising the existence of right (respectively left) adjoints in terms of a universal property.
Dependent types. Dependent product sets and dependent function sets as adjoint functors.
Equivalence of categories. Example: the category of I-indexed sets and functions is equivalent to the slice category Set/I.
Presheaves. The Yoneda Lemma. Categories of presheaves are cartesian closed.
Monads. Modelling notions of computation as monads. Moggi's computational lambda calculus.
Objectives
On completion of this module, students should:

be familiar with some of the basic notions of category theory and its connections with logic and programming language semantics
Assessment
a graded exercise sheet (25% of the final mark), and
a take-home test (75%)
Recommended reading
Awodey, S. (2010). Category theory. Oxford University Press (2nd ed.).

Crole, R. L. (1994). Categories for types. Cambridge University Press.

Lambek, J. and Scott, P. J. (1986). Introduction to higher order categorical logic. Cambridge University Press.

Pitts, A. M. (2000). Categorical Logic. Chapter 2 of S. Abramsky, D. M. Gabbay and T. S. E. Maibaum (Eds) Handbook of Logic in Computer Science, Volume 5. Oxford University Press. (Draft copy available here.)

Class Size
This module can accommodate upto 15 Part II students plus 15 MPhil / Part III students.

Pre-registration evaluation form
Students who are interested in taking this module will be asked to complete a pre-registration evaluation form to determine whether they have enough mathematical background to take the module.

This will take place during the week Monday 12 August - Friday 16 August.

Coursework
The coursework in this module will consist of a graded exercise sheet.

Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take 'Catgeory Theory' as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

DIGITAL MONEY AND DECENTRALISED FINANCE

Aims

Our society is evolving towards digital payments and the elimination of physical cash. Digital alternatives to cash require trade-offs between various properties including unforgeability, traceability, divisibility, transferability without intermediaries, privacy, redeemability, control over the money supply and many more, often in conflict with each other. Since the 1980s, cryptographers have proposed a variety of clever technical solutions addressing some of these issues but it is only with the appearance of Bitcoin, blockchain and a plethora of copycat cryptocurrencies that a new asset class has now emerged, worth in excess of two trillion dollars (although plagued by extreme volatility). Meanwhile, most major countries have been planning the introduction of Central Bank Digital Currencies in an attempt to retain control. This seminar-style interactive module encourages the students to understand what's happening, what the underlying problems and opportunities are (technical, social and economic) and where we should go from here. We are at a historical turning point where an enterprising candidate might disrupt the status quo and truly make a difference.

Syllabus

Understanding money: from gold standard to digital cash
Intro to the Decentralised Finance (DeFi) ecosystem: Bitcoin, blockchain, wallets, smart contracts
Stablecoins and Central Bank Digital Currencies
Decentralised Autonomous Organisations and Automated Market Makers
Yield Farming, Perpetual Futures and Maximal Extractable Value
Web3, Non Fungible Tokens
Cryptocurrency crashes; crimes facilitated by cryptocurrencies
New areas of development. Where from here?
The module is structured as a reading club with a high degree of interactivity. Aside from the first and last session, which are treated differently, the regular two-hour sessions have the following structure (not necessarily in this order).

Two half-hour debates on each of two papers
Two rapid-fire presentations on open questions previously assigned by the lecturers.
Half an hour of presentation by the lecturers.
About ten minutes of buffer that can account for switchover time or be absorbed into the lecturers' presentation.
The first session is scene-setting by the lecturers, with no student presentations. The last session has no paper debates, only one rapid-fire presentation on an open question from each of the students.

Objectives

On completion of this module, students will gain an in-depth understanding of digital money alternatives as well as many modern applications and components of Decentralised Finance. By comparing DeFi with current processes in traditional banking, students will be able to discuss the merits and limitations of these new technologies as well as to identify potential new areas of development.

Through having to write, present and defend very brief extended abstracts, the students will improve their communication skills and in particular the ability to distil and convey the most important points forcefully and concisely.

Assessment

The module is an interactive reading club. Each student produces four output triplets throughout the course. Each output triplet consists of three parts: a 1200-word essay, a 200-word extended abstract and a 7-minute presentation.

The essay and abstract must be submitted in advance, using the LaTeX templates provided. The abstract must consist of full sentences, not bullet points, and must fit on a single slide. During the presentation, the slide with the abstract will be projected on screen as the only visual aid; the essay will not be accessible to either presenter or audience. Reading out the slide verbatim will obviously not count as a great presentation. Presenters must be able to expand and defend their ideas live, and answer questions from lecturers and audience.

The essays, the abstracts and the presentations are graded separately, each out of 10, and assigned weights of 60%, 20%, 20%, respectively, within the triplet.

In a regular 2-hour session (all but the first and last) there will be two paper slots and two investigation slots, plus a lecturer slot, described next.

Each paper slot (30 min) involves a “supporter” and a “dissenter” for the paper, each contributing one output triplet, and then a panel Q&A where all the other participants quiz the supporter and dissenter.

Each investigation slot (10 min) involves a single student contributing an output triplet on a given theme.

Roles, papers and themes are assigned by the lecturers the previous week.

In the first session there will be no student presentations, only lecturer-led exploration. In the last session there will be 12 investigation slots and no paper slots.

Overall, each student will be supporter once, dissenter once and investigator twice, for a total of 4 output triplets. The output triplets of the last session will be weighed twice as much as the others, i.e. 40% each, while the others 20% each.

Attendance is required at all sessions. Missing a session in which the candidate was due to present will result in the loss of the corresponding presentation marks, while the written work will still have to be submitted and will be marked as usual. Missing a session in which the candidate was not due to present will result in the loss of 3%.

Recommended reading

There isn't a textbook covering all the material in this course. The following textbook, also adopted by R47 Distributed Ledger Technologies (also freely available online), is a thorough introduction to Bitcoin and Blockchain, but we will complement it with research papers and industry white papers for each lecture.

Narayanan, A. , Bonneau, J., Felten, E., Miller, A. and Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.
(2016 Draft available here:
https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf)

DIGITAL SIGNAL PROCESSING

Aims

This course teaches basic signal-processing principles necessary to understand many modern high-tech systems, with examples from audio processing, image coding, radio communication, radar, and software-defined radio. Students will gain practical experience from numerical experiments in programming assignments (in Julia, MATLAB or NumPy).

Lectures

Signals and systems. Discrete sequences and systems: types and properties. Amplitude, phase, frequency, modulation, decibels, root-mean square. Linear time-invariant systems, convolution. Some examples from electronics, optics and acoustics.
Phasors. Eigen functions of linear time-invariant systems. Review of complex arithmetic. Phasors as orthogonal base functions.
Fourier transform. Forms and properties of the Fourier transform. Convolution theorem. Rect and sinc.
Dirac's delta function. Fourier representation of sine waves, impulse combs in the time and frequency domain. Amplitude-modulation in the frequency domain.
Discrete sequences and spectra. Sampling of continuous signals, periodic signals, aliasing, interpolation, sampling and reconstruction, sample-rate conversion, oversampling, spectral inversion.
Discrete Fourier transform. Continuous versus discrete Fourier transform, symmetry, linearity, FFT, real-valued FFT, FFT-based convolution, zero padding, FFT-based resampling, deconvolution exercise.
Spectral estimation. Short-time Fourier transform, leakage and scalloping phenomena, windowing, zero padding. Audio and voice examples. DTFM exercise.
Finite impulse-response filters. Properties of filters, implementation forms, window-based FIR design, use of frequency-inversion to obtain high-pass filters, use of modulation to obtain band-pass filters.
Infinite impulse-response filters. Sequences as polynomials, z-transform, zeros and poles, some analog IIR design techniques (Butterworth, Chebyshev I/II, elliptic filters, second-order cascade form).
Band-pass signals. Band-pass sampling and reconstruction, IQ up and down conversion, superheterodyne receivers, software-defined radio front-ends, IQ representation of AM and FM signals and their demodulation.
Digital communication. Pulse-amplitude modulation. Matched-filter detector. Pulse shapes, inter-symbol interference, equalization. IQ representation of ASK, BSK, PSK, QAM and FSK signals. [2 hours]
Random sequences and noise. Random variables, stationary and ergodic processes, autocorrelation, cross-correlation, deterministic cross-correlation sequences, filtered random sequences, white noise, periodic averaging.
Correlation coding. Entropy, delta coding, linear prediction, dependence versus correlation, random vectors, covariance, decorrelation, matrix diagonalization, eigen decomposition,

Karhunen-Loève transform, principal component analysis. Relation to orthogonal transform coding using fixed basis vectors, such as DCT.
Lossy versus lossless compression. What information is discarded by human senses and can be eliminated by encoders? Perceptual scales, audio masking, spatial resolution, colour coordinates, some demonstration experiments.
Quantization, image coding standards. Uniform and logarithmic quantization, A/μ-law coding, dithering, JPEG.
Objectives
apply basic properties of time-invariant linear systems;
understand sampling, aliasing, convolution, filtering, the pitfalls of spectral estimation;
explain the above in time and frequency domain representations;
use filter-design software;
visualize and discuss digital filters in the z-domain;
use the FFT for convolution, deconvolution, filtering;
implement, apply and evaluate simple DSP applications;
familiarity with a number of signal-processing concepts used in digital communication systems
Recommended reading
Lyons, R.G. (2010). Understanding digital signal processing. Prentice Hall (3rd ed.).
Oppenheim, A.V. and Schafer, R.W. (2007). Discrete-time digital signal processing. Prentice Hall (3rd ed.).
Stein, J. (2000). Digital signal processing – a computer science perspective. Wiley.

Class size
This module can accommodate a maximum of 24 students (16 Part II students and 8 MPhil students)

Assessment - Part II students
Three homework programming assignments, each comprising 20% of the mark
Written test, comprising 40% of the total mark.
Assessment - Part III and MPhil students
Three homework programming exercises, each comprising 10% of the mark.
Written test, comprising 20% of the total mark.
Individual implementation project with 8-page written report, topic agreed with lecturer, comprising 50% of the mark.
Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

INTRODUCTION TO COMPUTATIONAL SEMANTICS

Aims
This is a lecture-style course that introduces students to various aspects of the semantics of Natural Languages (mainly English):

Lexical Semantics, with an emphasis on theory and phenomenology
Compositional Semantics
Discourse and pragmatics-related aspects of semantics
Objectives
Give an operational definition of what is meant by “meaning” (for instance, above and beyond syntax);
Name the types of phenomena in language that require semantic consideration, in terms of lexical, compositional and discourse/pragmatic aspects, in other words, argue why semantics is important;
Demonstrate an understanding of the basics of various semantic representations, including logic-based and graph-based semantic representations, their properties, how they are used and why they are important, and how they are different from syntactic representations;
Know how such semantic representations are derived during or after parsing, and how they can be analysed and mapped to surface strings;
Understand applications of semantic representations e.g. reasoning, validation, and methods how these are approached.
When designing NL tasks that clearly require semantic processing (e.g. knowledge-based QA), to be aware of and reuse semantic representations and algorithms when designing the task, rather than reinventing the wheel.
Practical advantages of this course for NLP students
Knowledge of underlying semantic effects helps improve NLP evaluation, for instance by providing more meaningful error analysis. You will be able to link particular errors to design decisions inside your system.
You will learn methods for better benchmarking of your system, whatever the task may be.
Supervised ML systems (in particular black-box systems such as Deep Learning) are only as clever as the datasets they are based on. In this course, you will learn to design datasets so that they are harder to trick without real understanding, or critique existing datasets.
You will be able to design tests for ML systems that better pinpoint which aspects of language an end-to-end system has “understood”.
You will learn to detect ambiguity and ill-formed semantics in human-human communication. This can serve to write more clearly and logically.
You will learn about decomposing complex semantics-reliant tasks sensibly so that you can reuse the techniques underlying semantic analyzers in a modular way. In this way, rather than being forced to treat complex tasks in an end-to-end manner, you will be able to profit from partial explanations and a better error analysis already built into the system.
Syllabus
Overview of the course
Events and semantic role labelling

Referentiality and coreference resolution
Truth-conditional semantics
Graph-based meaning representation
Compositional semantics
Context-free Graph Rewriting
Surface realisation
Negation and psychological approach to semantics
Dynamic semantics
Gricean pragmatics
Vector space models
Cross-modality
Acquisition of semantics
Diachronic change of semantics
Summary of the course

Assessment
5 take-home exercises worth 20% each:

Take-home assignment 1 is given in Lecture 4 and due is Lecture 6

Students are given 10 English sentences and asked to provide their semantic analysis according to truth-conditions. Students will be expected to return 10 logical expressions as their answers. No word limit. Assessment criteria: correctness of the 10 logical expressions; 2 points for each logical expression.

Take-home assignment 2 is given in Lecture 7 and due is Lecture 9

Students are given 10 English sentences and asked to provide their syntactico-semantic derivations according to the compositionality principle. Students will be expected to return derivation graphs as their answers. No word limit. Assessment criteria: correctness of the 10 derivation graphs; 2 points for each derivation.

Take-home assignment 3 is given in Lecture 11 and due is Lecture 13

All students are assigned with a paper on modelling common ground in dialogue system. Students will receive related but different papers. Each student will write a review of their assigned paper, including a comprehensive summary and their own thoughts. Word limit: 1000 words. Assessment criteria: 15 points on whether a student understands the paper correctly; 5 points on whether a student is able to think critically.

Take-home assignment 4 is given in Lecture 13 and due is Lecture 15

All students are assigned with a paper on language-vision interaction. Students will receive related but different papers. Each student will write a review of their assigned paper, including a comprehensive summary and their own thoughts. Word limit: 1000 words. Assessment criteria: 15 points on whether a student understands the paper correctly; 5 points on whether a student is able to think critically.

Take-home assignment 5 is given in Lecture 14 and due is two weeks later.

Content: All students are assigned with a paper on bootstrapping language acquisition. All students will receive the same paper. Each student will write a review of the paper, including a comprehensive summary and their own thoughts. Word limit: 1000 words. Assessment criteria: 15 points on whether a student understands the paper correctly; 5 points on whether a student is able to think critically.

INTRODUCTION TO NATURAL LANGUAGE SYNTAX AND PARSING

Aims
This module aims to provide a brief introduction to linguistics for computer scientists and then goes on to cover some of the core tasks in natural language processing (NLP), focussing on statistical parsing of sentences to yield syntactic representations. We will look at how to evaluate parsers and see how well state-of-the-art tools perform given current techniques.

Syllabus
Linguistics for NLP focusing on morphology and syntax
Grammars and representations
Statistical and neural parsing
Treebanks and evaluation
Objectives
On completion of this module, students should:

understand the basic properties of human languages and be familiar with descriptive and theoretical frameworks for handling these properties;
understand the design of tools for NLP tasks such as parsing and be able to apply them to text and evaluate their performance;
Practical work
Week 1-7: There will be non-assessed practical exercises between sessions and during sessions.
Week 8: Download and apply two parsers to a designated text. Evaluate the performance of the tools quantitatively and qualitatively.
Assessment
There will be one presentation worth 10% of the final mark; presentation topics will be allocated at the start of term.
An assessed practical report of not more 8 pages in ACL conference format. It will contribute 90% of the final mark.
Recommended reading
Jurafsky, D. and Martin, J. (2008). Speech and Language Processing. Prentice-Hall (2nd ed.). (See also 3rd ed. available online.)

INTRODUCTION TO NETWORKING AND SYSTEMS MEASUREMENTS

Aims

Systems research refers to the study of a broad range of behaviours arising from complex system design, including: resource sharing and scheduling; interactions between hardware and software; network topology, protocol and device design and implementation; low-level operating systems; Interconnect, storage and more. This module will:

Teach performance characteristics and performance measurement methodology and practice through profiling experiments;
Expose students to real-world systems artefacts evident through different measurement tools;
Develop scientific writing skills through a series of laboratory reports;
Provide research skills for characterization and modelling of systems and networks using measurements.

Prerequisites

It is strongly recommended that students have previously (and successfully) completed an undergraduate networking course -- or have equivalent experience through project or open-source work.

Syllabus

Introduction to performance measurements, performance characteristics [1 lecture]
Performance measurements tools and techniques [2 lectures + 2 lab sessions]
Reproducible experiments [1 lecture + 1 lab session]
Common pitfalls in measurements [1 lecture]
Device and system characterisation [1 lecture + 2 lab sessions]

Objectives

On completion of this module, students should:

Describe the objectives of measurements, and what they can achieve;
Characterise and model a system using measurements;
Perform reproducible measurements experiments;
Evaluate the performance of a system using measurements;
Operate measurements tools and be aware of their limitations;
Detect anomalies in the network and avoid common measurements pitfalls;
Write system-style performance evaluations.

Practical work

Five 2-hour in-classroom labs will ask students to develop and use skills in performance measurements as applied to real-world systems and networking artefacts. Results from these labs (and follow-up work by students outside of the classroom) will by the primary input to lab reports.

The first three labs will provide an introduction and hands on experience with measurement tools and measurements methodologies, while the last two labs will focus on practical measurements and evaluation of specific platforms. Students may find it useful to work in pairs within the lab, but must prepare lab reports independently. The module lecturer will give a short introductory at the start of each lab, and instructors will be on-hand throughout labs to provide assistance.

Lab participation is not directly included in the final mark, but lab work is a key input to lab reports that are assessed. Guided lab experiments resulting in practical write ups.

Assessment
Each student will write two lab reports. The first lab report will summarise the experiments done in the first three labs (20%). The second will be a lab report (5000 words) summarising the evaluation of a device or a system (80%).

Recommended reading
The following list provides some background to the course materials, but is not mandatory. A reading list, including research papers, will be provided in the course materials.

George Varghese. Network algorithmics. Chapman and Hall/CRC, 2010.
Mark Crovella and Balachander Krishnamurthy. Internet measurement: infrastructure, traffic and applications. John Wiley and Sons, Inc., 2006.
Brendan Gregg. Systems Performance: Enterprise and the Cloud, Prentice Hall Press, Upper Saddle River, NJ, USA, October 2013.
Raj Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley - Interscience, New York, NY, USA, April 1991.

LARGE-SCALE DATA PROCESSING AND OPTIMISATION

Aims

This module provides an introduction to large-scale data processing, optimisation, and the impact on computer system's architecture. Large-scale distributed applications with high volume data processing such as training of machine learning will grow ever more in importance. Supporting the design and implementation of robust, secure, and heterogeneous large-scale distributed systems is essential. To deal with distributed systems with a large and complex parameter space, tuning and optimising computer systems is becoming an important and complex task, which also deals with the characteristics of input data and algorithms used in the applications. Algorithm designers are often unaware of the constraints imposed by systems and the best way to consider these when designing algorithms with massive volume of data. On the other hand, computer systems often miss advances in algorithm design that can be used to cut down processing time and scale up systems in terms of the size of the problem they can address. Integrating machine learning approaches (e.g. Bayesian Optimisation, Reinforcement Learning) for system optimisation will be explored in this course.

Syllabus

This course provides perspectives on large-scale data processing, including data-flow programming, graph data processing, probabilistic programming and computer system optimisation, especially using machine learning approaches, thus providing a solid basis to work on the next generation of distributed systems.

The module consists of 8 sessions, with 5 sessions on specific aspects of large-scale data processing research. Each session discusses 3-4 papers, led by the assigned students. One session is a hands-on tutorial on on dataflow programming fundamentals. The first session advises on how to read/review a paper together with a brief introduction on different perspectives in large-scale data
processing and optimisation. The last session is dedicated to the student presentation of opensource project studies.

Introduction to large-scale data processing and optimisation
Data flow programming: Map/Reduce to TensorFlow
Large-scale graph data processing: storage, processing model and parallel processing
Dataflow programming hands-on tutorial
Probabilistic Programming
Optimisations in ML Compiler
Optimisations of Computer systems using ML
Presentation of Open Source Project Study

Objectives

On completion of this module, students should:

Understand key concepts of scalable data processing approaches in future computer systems.

Obtain a clear understanding of building distributed systems using data centric programming and large-scale data processing.
Understand a large and complex parameter space in computer system's optimisation and applicability of Machine Learning approach.
Coursework
Reading Club:
The preparation for the reading club will involve 1-3 papers every week. At each session, around 3-4 papers are selected under the given topic, and the students present their review work.
Hands-on tutorial session of data flow programming including writing an application of processing streaming in Twitter data and/or Deep Neural Networks using Google TensorFlow using cluster computing.
Reports
The following three reports are required, which could be extended from the assignment of the reading club, within the scope of data centric systems.

Review report on a full length paper (max 1800 words)
Describe the contribution of the paper in depth with criticisms
Crystallise the significant novelty in contrast to other related work
Suggestions for future work
Survey report on sub-topic in large-scale data processing and optimisation (max 2000 words)
Pick up to 5 papers as core papers in the survey scope
Read the above and expand reading through related work
Comprehend the view and finish an own survey paper
Project study and exploration of a prototype (max 2500 words)
What is the significance of the project in the research domain?
Compare with similar and succeeding projects
Demonstrate the project by exploring its prototype
Reports 1 should be handed in by the end of 5th week; Report 2 in the 8th week and Report 3 on the first day of Lent Term.

Assessment
The final grade for the course will be provided as a percentage, and the assessment will consist of two parts:

25%: for reading club (participation, presentation)
75%: for the three reports:
15%: Intensive review report
25%: Survey report
35%: Project study
Recommended reading
M. Abadi et al. TensorFlow: A System for Large-Scale Machine Learning, OSDI, 2016.
D. Aken et al.: Automatic Database Management System Tuning Through Large-scale Machine Learning, SIGMOD, 2017.

J. Ansel et al. Opentuner: an extensible framework for program autotuning. PACT, 2014.
V. Dalibard, M. Schaarschmidt, E. Yoneki. BOAT: Building Auto-Tuners with Structured Bayesian Optimization, WWW, 2017.
J. Dean et al. Large scale distributed deep networks. NIPS, 2012.
G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, G. Czajkowski. Pregel: A System for Large-Scale Graph Processing, SIGMOD, 2010.
A. Mirhoseini et al. Device Placement Optimization with Reinforcement Learning, ICML, 2017.
D. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, M. Abadi. Naiad: A Timely Dataflow System, SOSP, 2013.
M. Schaarschmidt, S. Mika, K. Fricke and E. Yoneki: RLgraph: Modular Computation Graphs for Deep Reinforcement Learning, SysML, 2019.
Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, A. Aiken: TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions: SOSP, 2019.
H. Mao et al.: Park: An Open Platform for Learning-Augmented Computer Systems, OpenReview, 2019.
J. Shao, et al.: Tensor Program Optimization with Probabilistic Programs, NeurIPS, 2022.
A complete list can be found on the course material web page. See also 2023-2024 course material on the previous course Large-Scale Data Processing and Optimisation.

MACHINE LEARNING AND THE PHYSICAL WORLD
Aims
The module “Machine Learning and the Physical World” is focused on machine learning systems that interact directly with the real world. Building artificial systems that interact with the physical world have significantly different challenges compared to the purely digital domain. In the real world data is scares, often uncertain and decisions can have costly and irreversible consequences. However, we also have the benefit of centuries of scientific knowledge that we can draw from. This module will provide the methodological background to machine learning applied in this scenario. We will study how we can build models with a principled treatment of uncertainty, allowing us to leverage prior knowledge and provide decisions that can be interrogated.

There are three principle points about machine learning in the real world that will concern us.

We often have a mechanistic understanding of the real world which we should be able to bootstrap to make decisions. For example, equations from physics or an understanding of economics.
Real world decisions have consequences which may have costs, and often these cost functions need to be assimilated into our machine learning system.
The real world is surprising, it does things that you do not expect and accounting for these challenges requires us to build more robust and or interpretable systems.
Decision making in the real world hasn’t begun only with the advent of machine learning technologies. There are other domains which take these areas seriously, physics, environmental scientists, econometricians, statisticians, operational researchers. This course identifies how machine learning can contribute and become a tool within these fields. It will equip you with an understanding of methodologies based on uncertainty and decision making functions for delivering on these challenges.

Objectives
You will gain detailed knowledge of

surrogate models and uncertainty
surrogate-based optimization
sensitivity analysis
experimental design
You will gain knowledge of

counterfactual analysis
surrogate-based quadrature
Schedule
Week 1: Introduction to the unit and foundation of probabilistic modelling

Week 2: Gaussian processes and probablistic inference

Week 3: Simulation and Sequential decision making under uncertainty

Week 4: Emulation and Experimental Design

Week 5: Sensitivity Analysis and Multifidelity Modelling

Week 6-8: Case studies of applications and Projects

Practical work
During the first five weeks of the unit we will provide a weekly worksheet that will focus on implementation and practical exploration of the material covered in the lectures. The worksheets will allow you to build up a these methods without relying on extensive external libraries. You are free to use any programming language of choice however we highly recommended the use of =Python=.

Assessment
Two individual tasks (15% each)
Group Project (70%). Each group will work on an application of uncertainty that covers the material of the first 5 weeks of lectures in the unit. Each group will submit a report which will form the basis of the assessment. In addition to the report each group will also attend a short oral examination based on the material covered both in the report and the taught material.
Recommended Reading
Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. MIT Press

Bishop, C. (2006). Pattern recognition and machine learning. Springer.
https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

Laplace, P. S. (1902). A Philosophical Essay on Probabilities. John Wiley & Sons.
https://archive.org/details/philosophicaless00lapliala

MACHINE VISUAL PERCEPTION

Aims

This course aims at introducing the theoretical foundations and practical techniques for machine perception, the capability of computers to interpret data resulting from sensor measurements. The course will teach the fundamentals and modern techniques of machine perception, i.e. reconstructing the real world starting from sensor measurements with a focus on machine perception for visual data. The topics covered will be image/geometry representations for machine perception, semantic segmentation, object detection and recognition, geometry capture, appearance modeling and acquisition, motion detection and estimation, human-in-the-loop machine perception, select topics in applied machine perception.

Machine perception/computer vision is a rapidly expanding area of research with real-world applications. An understanding of machine perception is also important for robotics, interactive graphics (especially AR/VR), applied machine learning, and several other fields and industries. This course will provide a fundamental understanding of and practical experience with the relevant techniques.

Learning outcomes

Students will understand the theoretical underpinnings of the modern machine perception techniques for reconstructing models of reality starting from an incomplete and imperfect view of reality.

Students will be able to apply machine perception theory to solve practical problems, e.g. classification of images, geometry capture.

Students will gain an understanding of which machine perception techniques are appropriate for different tasks and scenarios.

Students will have hands-on experience with some of these techniques via developing a functional machine perception system in their projects.

Students will have practical experience with the current prominent machine perception frameworks.

Syllabus

The fundamentals of machine learning for machine perception

Deep neural networks and frameworks for machine perception

Semantic segmentation of objects and humans

Object detection and recognition

Motion estimation, tracking and recognition

3D geometry capture

Appearance modeling and acquisition

Select topics in applied machine perception

Assessment

A practical exercise, worth 20% of the mark. This will cover the basics and theory of machine perception and some of the practical techniques the students will likely use in their projects. This is individual work. No GPU hours will be needed for the practical work.

A machine perception project worth 80% of the marks:

Course projects will be selected by the students following the possible project themes proposed by the lecturer, and will be checked by the lecturer for appropriateness.
The students will form groups of 2-3 to design, implement, report, and present a project to tackle a given task in machine perception.
The final mark will be composed of an implementation/report mark (60%) and a presentation mark (20%). Each team member will be evaluated based on her/his contribution.
Each project will have extensions to be completed only by the ACS students. Each student will write a different part of the report, whose author will be clearly marked. Each student will further summarise her/his contributions to the project in the same report.
Recommended Reading List
Computer Vision: Algorithms and Applications, Richard Szeliski, Springer, 2010.
Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, MIT Press, 2016.
Machine Learning and Visual Perception, Baochang Zhang, De Gruyter, 2020.
Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

MOBILE, WEARABLE SYSTEMS AND MACHINE LEARNING

Aims
This module aims to introduce the latest research advancements in mobile systems and mobile data machine learning, spanning a range of domains including systems, data gathering, analytics and machine learning, on device machine learning and applications such as health, transportation, behavior monitoring, cyber-physical systems, autonomous vehicles, drones. The course will cover current and seminal research papers in the area of research.

Syllabus
The course will consist of one introductory lecture, seven two-hour, and one three-hour, sessions covering a variety of topics roughly including the following material (some variation in the topics might happen from year to year):

System, Energy and Security
Backscatter Communication, Battery Free and Energy Harvesting Devices
New Sensing Modalities
Machine Learning on Wearable Data
On Device Machine Learning
Mobile and Wearable Health
Mobile and Wearable Systems of Sustainability
Each week, three class participants will be assigned to introduce assigned three papers via 20-minute presentations, conference-style and highlighting critically its features. Each presentation will be followed by 10 minutes of questions. This will be followed by 10 minutes of general discussion. Slides will be used for presentation.

Students will give one or more presentations each term. Each student will submit a paper review each week for one of the three papers presented except for the week they will be presenting slides. Each review will follow a template and be up to 1,000 words. Each review will receive a maximum of 10 points. As a result, each student will produce 6-7 reviews and at least one presentation, probably two. All participants are expected to attend and participate in every class; the instructor must be notified of any absences in advance.

Objectives
On completion of this module students should have an understanding of the recent key research in mobile and sensor systems and mobile analytics as well as an improved critical thinking over research papers.

Assessment
Aggregate mark for 7 assignments from 6-7 reports and 1-2 presentations. Each report or presentation will contribute one seventh of the course mark.
A tick for presence and participation to each class will also be awarded.
Recommended reading

Readings from the most recent conferences such as AAAI, ACM KDD, ACM MobiCom, ACM MobiSys, ACM SenSys, ACM UbiComp, ICLR, ICML Neurips, and WWW pertinent to mobile systems and data.

NETWORK ARCHITECTURES

Aims

This module aims to provide the world with more network architects. The 2011-2012 version was oriented around the evolution of IP to support new services like multicast, mobility, multihoming, pub/sub and, in general, data oriented networking. The course is a paper reading which puts the onus on the student to do the work.

Syllabus

IPng [2 lectures, Jon Crowcroft]
New Architectures [2 lectures, Jon Crowcroft]
Multicast [2 lectures, Jon Crowcroft]
Content Distribution and Content Centric Networks [2 lectures, Jon Crowcroft]
Resource Pooling [2 lectures, Jon Crowcroft]
Green Networking [2 lectures, Jon Crowcroft]
Alternative Router Implementions [2 lectures, Jon Crowcroft]
Data Center Networks [2 Lectures, Jon Crowcroft]

Objectives

On completion of this module, students should be able to:

contribute to new network system designs;
engineer evolutionary changes in network systems;
identify and repair architectural design flaws in networked systems;
see that there are no perfect solutions (aside from academic ones) for routing, addressing, naming;
understand tradeoffs in modularisation and other pressures on clean software systems implementation, and see how the world is changing the proper choices in protocol layering, or non layered or cross-layered.

Coursework

Assessment is through three graded essays (each chosen individually from a number of suggested or student-chosen topics), as follows:

Analysis of two different architectures for a particular scenario in terms of cost/performance tradeoffs for some functionality and design dimension, for example:
ATM – e.g. for hardware versus software tradeoff
IP – e.g. for mobility, multi-homing, multicast, multipath
3GPP – e.g. for plain complexity versus complicatedness
A discursive essay on a specific communications systems component, in a particular context, such as ad hoc routing, or wireless sensor networks.
A bespoke network design for a narrow, well specified specialised target scenario, for example:
A customer baggage tracking network for an airport.
in-flight entertainment system.
in-car network for monitoring and control.
inter-car sensor/control network for automatic highways.

Practical work

This course does not feature any implementation work due to time constraints.

Assessment
Three 1,200-word essays (worth 25% each), and
an annotated bibliography (25%).
Recommended reading
Pre-course reading:

Keshav, S. (1997). An engineering approach to computer networking. Addison-Wesley (1st ed.). ISBN 0201634422
Peterson, L.L. and Davie, B.S. (2007). Computer networks: a systems approach. Morgan Kaufmann (4th ed.).

Design patterns:

Day, John (2007). Patterns in network architecture: a return to fundamentals. Prentice Hall.

Example systems:

Krishnamurthy, B. and Rexford, J. (2001). Web protocols and practice: HTTP/1.1, Networking protocols, caching, and traffic measurement. Addison-Wesley.

Economics and networks:

Frank, Robert H. (2008). The economic naturalist: why economics explains almost everything.

Papers:

Certainly, a collection of papers (see ACM CCR which publishes notable network researchers' favourite ten papers every 6 months or so).

OVERVIEW OF NATURAL LANGUAGE PROCESSING

Aims
This course introduces the fundamental techniques of natural language processing. It aims to explain the potential and the main limitations of these techniques. Some current research issues are introduced and some current and potential applications discussed and evaluated. Students will also be introduced to practical experimentation in natural language processing.

Lectures
Overview. Brief history of NLP research, some current applications, components of NLP systems.
Morphology and Finite State Techniques. Morphology in different languages, importance of morphological analysis in NLP, finite-state techniques in NLP.
Part-of-Speech Tagging and Log-Linear Models. Lexical categories, word tagging, corpora and annotations, empirical evaluation.
Phrase Structure and Structure Prediction. Phrase structures, structured prediction, context-free grammars, weights and probabilities. Some limitations of context-free grammars.
Dependency Parsing. Dependency structure, grammar-free parsing, incremental processing.
Gradient Descent and Neural Nets. Parameter optimisation by gradient descent. Non-linear functions with neural network layers. Log-linear model as softmax layer. Current findings of Neural NLP.
Word representations. Representing words with vectors, count-based and prediction-based approaches, similarity metrics.
Recurrent Neural Networks. Modelling sequences, parameter sharing in recurrent neural networks, neural language models, word prediction.
Compositional Semantics. Logical representations, compositional semantics, lambda calculus, inference and robust entailment.
Lexical Semantics. Semantic relations, WordNet, word senses.
Discourse. Discourse relations, anaphora resolution, summarization.
Natural Language Generation. Challenges of natural language generation (NLG), tasks in NLG, surface realisation.
Practical and assignments. Students will build a natural language processing system which will be trained and evaluated on supplied data. The system will be built from existing components, but students will be expected to compare approaches and some programming will be required for this. Several assignments will be set during the practicals for assessment.
Objectives
By the end of the course students should:

be able to discuss the current and likely future performance of several NLP applications;
be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing, parsing, word sense disambiguation etc.;
understand how these techniques draw on and relate to other areas of computer science.
Recommended reading

Jurafsky, D. & Martin, J. (2023). Speech and language processing. Prentice Hall (3rd ed. draft, online).

Assessment - Part II Students
Assignment 1 - 10% of marks
Assignment 2 - 60% of marks
Assignment 3 - 30% of marks
Coursework
Submit work for 3 assignments as part of practical sessions on information extraction.

These include an annotation exercise, a feature-based classifier along with code repository and documentation, and a 4,000-word report on results and analysis from an extended information extraction experiment.

Practical work
Students will build a natural language processing system which will be trained and evaluated on supplied data. The system will be built from existing components, but students will be expected to compare approaches and some programming will be required for this.

Assessment - Part III and MPhil Students
Assessment will be based on the practicals:

First practical exercise (10%, ticked)
Second practical exercise (25%, code repository or notebook with documentation)
Final report (65%, 4,000 words, excluding references)
Further Information
Although the lectures don't assume any exposure to linguistics, the course will be easier to follow if students have some understanding of basic linguistic concepts. The following may be useful for this: The Internet Grammar of English

Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

Students from other departments may attend the lectures for this module if space allows. However students wanting to take it for credit will need to make arrangements for assessment within their own department.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

PRACTICAL RESEARCH IN HUMAN-CENTRED AI

Aims

This is an advanced course in human-computer interaction, with a specialist focus on intelligent user interfaces and interaction with machine-learning and artificial intelligence technologies. The format will be largely Practical, with students carrying out a mini-project involving empirical research investigation. These studies will investigate human interaction with some kind of model-based system for planning, decision-making, automation etc. Possible study formats might include: System evaluation, Field observation, Hypothesis testing experiment, Design intervention or Corpus analysis, following set examples from recent research publications. Project work will be formally evaluated through a report and presentation.

Lectures

(note that Lectures 2-7 also include one hour class discussion of practical work)

- Current research themes in intelligent user interfaces
- Program synthesis
- Mixed initiative interaction
- Interpretability / explainable AI
- Labelling as a fundamental problem
- Machine learning risks and bias
- Visualisation and visual analytics
- Student research presentations

Objectives

By the end of the course students should:

- be familiar with current state of the art in intelligent interactive systems
- understand the human factors that are most critical in the design of such systems
- be able to evaluate evidence for and against the utility of novel systems
- have experience of conducting user studies meeting the quality criteria of this field
- be able to write up and present user research in a professional manner

Class Size

This module can accommodate upto 20 Part II, Part III and MPhil students.

Recommended reading

Brad A. Myers and Richard McDaniel (2000). Demonstrational Interfaces: Sometimes You Need a Little Intelligence, Sometimes You Need a Lot.

Alan Blackwell (2024). Moral Codes: Designing alternatives to AI

Assessment - Part II Students

The format will be largely practical, with students carrying out an individual mini-project involving empirical research investigation.

Assignment 1: six incremental submissions which together contribute 20% to the final module mark.

Assignment 2: Final report - 80% of the final module mark

Assessment - MPhil / Part III Students
There will be a minor assessment component (20%) in which students compile a reflective diary throughout the term, reporting on the weekly sessions. Diary entries should include citations to any key references, notes of possible further reading, summary of key points, questions relevant to the personal project, and points of interest noted in relation to the work of other students.

The major assessment component (80%) will involve a report on research findings, in the style of a submission to the ACM CHI or IUI conferences. This work will be submitted incrementally through the term, in order that feedback can be provided before final assessment of the full report. Phased submissions will cover the following aspects of the empirical study:

Research question
Method
Literature Review
Introduction
Results
Discussion / Conclusion
Feedback on each phased submission will include an indicative mark for guidance, but with the understanding that the final grade will be based on the final delivered report, and that this may go up (or possibly down), depending on how well the student responds to earlier feedback.

Reflective diary entries will also be assessed, but graded at a relatively coarse granularity corresponding to the ACS grading bands, and with minimal written feedback. Informal and generic feedback will be offered verbally in class, and also potentially supplemented with peer assessment.

Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

PRINCIPLES OF MACHINE LEARNING SYSTEMS

Prerequisites: It is recommended that students have successfully completed introductory courses, at an undergraduate level, in: 1) operating systems, 2) computer architecture, and 3) machine learning. In addition, this course will heavily focus on deep neural network methodologies and assume familiarity with common neural architectures and related algorithms. If these topics were not covered within the machine learning course taken, students should supplement by reviewing material in a book like: "Dive into Deep Learning". Finally, students are assumed to be comfortable with programming in Python.
Moodle, timetable

Aims
This course will examine the emerging principles and methodologies that underpin scalable and efficient machine learning systems. Primarily, the course will focus on an exciting cross-section of algorithms and system techniques that are used to support the training and inference of machine learning models under a spectrum of computing systems that range from constrained embedded systems up to large-scale distributed systems. It will also touch upon the new engineering practices that are developing in support of such systems at scale. When needed to appreciate issues of scalability and efficiency, the course will drill down to certain aspects of computer architecture, systems software and distributed systems and explore how these interact with the usage and deployment of state-of-the-art machine learning.

Syllabus
Topics covered may include the following, with confirmation a month before the course begins:

System Performance Trade-offs
Distributed Learning Algorithms
Model Compression
Deep Learning Compilers
Frameworks and Run-times
Scalable Inference Serving
Development Practices
Automated Machine Learning
Federated Learning
Primarily, topics are covered with conventional lectures. However, where appropriate, material will be delivered through hands-on lab tutorials. Lab tutorials will make use of hardware including ARM microcontrollers and multi-GPU machines to explore forms of efficient machine learning (any necessary equipment will be provided to students)

Assessment
Each student will be assessed on 3 labs which will be worth 30% of their grade. They will also undertake a written project report which will be worth 70% of the grade. This report will detail an investigation into a particular aspect of machine learning systems, this report will be made available publicly.

PROOF ASSISTANTS

Aims

This module introduces students to interactive theorem proving using Isabelle and Coq. It includes techniques for specifying formal models of software and hardware systems and for deriving properties of these models.

Syllabus

Introduction to proof assistants and logic.
Reasoning in predicate logic and typed set theory.
Reasoning in dependent type theory.
Inductive definitions and recursive functions: modelling them in logic, reasoning about them.
Modelling operational semantics definitions and proving properties.

Objectives

On completion of this module, students should:

possess good skills in the use of Isabelle and Coq;
be able to specify inductive definitions and perform proofs by induction;
be able to formalise and reason about a variety of specifications in a proof assistant.

Coursework

Practical sessions will allow students to develop skills. Some of the exercises will serve as the basis for assessment.

Assessment

Assessment will be based on two marked assignments (due in weeks 5 and 8) with students performing small formalisation and verification projects in a proof assistant together with a write-up explaining their work (word limit 2,500 per project for the write-up). Each of the assignments will be worth 100 marks, distributed as follows:

50 marks for completing basic formalisation and verification tasks assessing grasp of the material taught in the lecture. Students will submit their work as theory files for either the Isabelle or Coq proof assistant, and they will be assessed for correctness and completeness of the specifications and proofs.
20 marks for completing designated more challenging tasks, requiring the use of advanced techniques or creative proof strategies.
30 marks for a clear write-up explaining the design decisions made during the formalisation and the strategy used for the proofs, where 10 of these marks will be reserved for write-ups of exceptional quality, e.g. demonstrating particular insight.
The main tasks in the assignments will be designed to assess the student's proficiency with the basic material and techniques taught in the lectures and the practical sessions, while the more challenging tasks will give exceptional students the opportunity to earn distinction marks

Recommended reading

Nipkow, T., Klein, G. (2014). Concrete Semantics with Isabelle/HOL. (The first part of this book, “Programming and Proving in Isabelle/HOL”, comes with the Isabelle distribution.)

Nipkow, T., Paulson, L.C. and Wenzel, M. (2002). A proof assistant for higher-order logic. Springer LNCS 2283.

The Software Foundations series of books, in particular the first two:

Pierce, B., Azevedo de Amorim, A., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., Sjöberg, V., Yorgey, B. (2023). Logical Foundations
Pierce, B., Azevedo de Amorim, A., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., Sjöberg, V., Tolmach, A., Yorgey, B. (2023). Programming Language Foundations
Chlipala, A. (2022). Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant. MIT Press.

Sergey, I. (2014). Programs and Proofs: Mechanizing Mathematics with Dependent Types.

All of these are freely available online.

QUANTUM ALGORITHMS AND COMPLEXITY

Aims
This module is a research-focused introduction to the theory of quantum computing. The aim is to prepare the students to conduct research in quantum algorithms and quantum complexity theory.

Syllabus
Topics will vary from year to year, based on developments in cutting edge research.
Representative topics include:

Quantum algorithms:

Quantum learning theory
Quantum property testing
Shadow tomography
Quantum walks
Quantum state/unitary synthesis.
Structure vs randomness in quantum algorithms
Quantum complexity theory:

The quantum PCP conjecture
Entanglement and MIP
State complexity, AdS/CFT, and quantum gravity
Quantum locally testable codes
Pseudorandom states and unitaries
Quantum zero-knowledge proofs
Objectives
On completion of this module, students should:

be familiar with contemporary quantum algorithms
develop an understanding of the power and limitations of quantum computation
conduct research in quantum algorithms and complexity theory
Assessment
Mini research project (70%)
Presentation (20%)
Attendance and participation (10%)
Timelines for the assignment submissions:

submit questions/observations on a weekly basis (i.e., Weeks 2-8)
deliver talks in Weeks 5-8
submit their mini-project at the end of term
Recommended reading material and resources
“Quantum Computing Since Democritus” by Scott Aaronson

“Introduction to Quantum Information Science” by Scott Aaronson

“Quantum Computing: Lecture Notes” by Ronald de Wolf

“Quantum Computation and Quantum Information” by Nielsen and Chuang

UNDERSTANDING QUANTUM ARCHITECTURE

Prerequisites: Requires introductory linear algebra - concepts such as eigenvalues, Hermitian matrices, unitary matrices. Taking the Part II Quantum Computing course (or similar) is helpful but not required. Familiarity with computer architecture and compilers is helpful.
Moodle, timetable

Aims

This course covers the architecture of a practical-scale quantum computer. We will examine the resource requirements of practical quantum applications, understand the different layers of the quantum stack, the techniques used in these layers and examine how these layers come together to enable practical quantum advantage over classical computing.

Syllabus

The course has two parts: a series of lectures to cover important aspects of the quantum stack and a set of student presentations. The following are a list of representative topics:

- Basics of quantum computing
- The fault-tolerant quantum stack
- Compilation
- Instruction sets
- Implementations of quantum error correction
- Implementation of magic state distillation
- Resource estimation

Student presentations will be based on a reading list of important papers in quantum architecture.

Objectives

At the end of the course, students will have a broad understanding of the quantum computing stack. They will understand how major qubit technologies work, design challenges in real quantum hardware, how quantum applications are mapped to a system and the importance of quantum error correction for scalability.

Assessment

Seminar presentation: 20%
Read one paper from a provided reading list
Prepare a 20 minute presentation on it + 10 minutes of answering questions.
The student presentation should be similar to a conference presentation - convey the problem, what are prior solutions, what did the paper do, what are the results, what are future directions
For the 20% of the score, the score will be split as 15% and 5%. 15% based on how well the student understands and explains the paper to the rest of the audience. 5% based on the Q&A session.

Course project: 80% (split across a proposal, mid-term report, final report)
A. Research proposal - at least 500 words (10% of total)
B. Mid-term report - at least 1000 words (including aspects like the research problem, literature review, what questions will be evaluated, any early ideas or methods (20% of total)
C. Final report - up to 4 pages double column in a conference paper style with 3000-4000 words. In addition to the mid term report, should include aspects like results and future directions. (50% of the total)
D. Report on contributions (in case of group work by 2 students) - 1 paragraph on individual contribution of the student. 1 paragraph on teammate's contribution.
Students may work alone in the project, in which case they need only components A-C. Students may work in pairs of two, but they should in addition individually submit D. The instructor may conduct a short viva in case contributions from both team members are not clear or imbalanced.
Recommended reading
Nielsen M.A., Chuang I.L. (2010). Quantum Computation and Quantum Information. Cambridge University Press.

Mermin N.D. (2007). Quantum Computer Science: An Introduction. Cambridge University Press.

Assessing requirements to scale to practical quantum advantage (2022) Beverland et al.

8th Semester
ADVANCED TOPICS IN CATEGORY THEORY

Teaching
The teaching style will be lecture-based, but supported by a practical component where students will learn to use a proof assistant for higher category theory, and build a small portfolio of proofs. Towards the end of the course we will explore some of the exciting computer science research literature on monoidal and higher categories, and students will choose a paper and present it to the class.

Aims
The module will introduce advanced topics in category theory. The aim is to train students to engage and start modern research on the mathematical foundations of higher categories, the graphical calculus, monoids and representations, type theories, and their applications in theoretical computer science, both classical and quantum.

Objectives
On completion of this module, students should:

Be familiar with the techniques of compositional category theory.
Have a strong understanding of basic categorical semantic models.
Begun exploring current research in monoidal categories and higher structures.
Syllabus
Part 1, lecture course:
The first part of the course introduces concepts from monoidal categories and higher categories, and explores their application in computer science.
- Monoidal categories and the graphical calculus
- The proof assistant homotopy.io
- Coherence theorems and higher category theory
- Linearity, superposition, duality, quantum entanglement
- Monoids, Frobenius algebras and bialgebras
- Type theory for higher category theory

Part 2, exploring the research frontier:
In the second part of the course, students choose a research paper to study, and give a presentation to the class.
There is a nice varied literature related to the topics of the course, and the lecturer will supply a list of suggested papers.

Classes
There will be three exercise sheets for homework, with accompanying classes by a teaching assistant to go over them.

Assessment
Problem sheets (50%)
Class presentation (20%)
Practical portfolio (30%)
Reading List
Chris Heunen and Jamie Vicary, “Category for Quantum Theory: An Introduction”, Oxford University Press

ADVANCED TOPICS IN COMPUTER SYSTEMS

Aims
This module will attempt to provide an overview of “systems” research. This is a very broad field which has existed for over 50 years and which has historically included areas such as operating systems, database systems, file systems, distributed systems and networking, to name but a few. The course will thus necessarily cover only a tiny subset of the field.

Many good ideas in systems research are the result of discussing and debating previous work. A primary aim of this course therefore will be to educate students in the art of critical thinking: the ability to argue for and/or against a particular approach or idea. This will be done by having students read and critique a set of papers each week. In addition, each week will include presentations from a number of participants which aim to advocate or criticise each piece of work.

Syllabus
The syllabus for this course will vary from year to year so as to cover a mixture of older and more contemporary systems papers. Contemporary papers will be generally selected from the past 5 years, primarily drawn from high quality conferences such as SOSP, OSDI, ASPLOS, FAST, NSDI and EuroSys. Example topics might include:

Systems Research and System Design
OS Structure and Virtual Memory
Virtualisation
Consensus
Scheduling
Privacy
Data Intensive Computing
Bugs
The reading each week will involve a load equivalent to 3 full length papers. Students will be expected to read these in detail and prepare a written summary and review. In addition, each week will contain one or more short presentations by students for each paper. The types of presentation will include:

Overview: a balanced presentation of the paper, covering both positive and negative aspects.
Advocacy: a positive spin on the paper, aiming to convince others of its value.
Criticism: a negative take on the paper, focusing on its weak spots and omissions.
These presentation roles will be assigned in advance, regardless of the soi disant absolute merit of the paper or the preference of the student. Furthermore, all students – regardless of any assigned presentation role in a given week – will be expected to participate in the class by asking questions and generally entering into the debate.

Objectives

On completion of this module students should have a broad understanding of some key papers and concepts in computer systems research, as well as an appreciation of how to argue for or against any particular idea.

Coursework and practical work
Coursework will be the production of the weekly paper reviews. Practical work will be presenting papers as appropriate, as well as ongoing participation in the class.

Assessment
Assessment consists of:

One essay per week for 7 weeks (10% each)
Presentation (20%)
Participation in class over the term (10%)
Recommended reading
Most of the reading for this course will be in the form of the selected papers each week. However, the following may be useful background reading to refresh your knowledge from undergraduate courses:

Silberschatz, A., Peterson, J.L. and Galvin, P.C. (2005). Operating systems concepts. Addison-Wesley (7th ed.).

Tanenbaum, A.S. (2008). Modern Operating Systems. Prentice-Hall (3rd ed.).

Bacon, J. and Harris, T. (2003). Operating systems. Addison-Wesley (3rd ed.).

Anderson, T. and Dahlin, M. (2014). Operating Systems: Principles and Practice. Recursive Books (2nd ed.).

Hennessy, J. and Patterson, D. (2006). Computer architecture: a quantitative approach. Elsevier (4th ed.). ISBN 978-0-12-370490-0.

Kleppmann M (2016) Designing Data-Intensive Applications, O'Reilly (1st ed.)

ADVANCED TOPICS IN MACHINE LEARNING
Aims
This course explores current research topics in machine learning in sufficient depth that, at the end of the course, participants will be in a position to contribute to research on their chosen topics. Each topic will be introduced with a lecture which, building on the material covered in the prerequisite courses, will make the current research literature accessible. Each lecture will be followed by up to three sessions which will typically be run as a reading group with student presentations on recent papers from the literature followed by a discussion, or a practical, or similar.

Structure
Each student will attend 3 topics and each topic's sessions will be spread over 5 contact hours. Students will be expected to undertake readings for their selected topics. There will be some group work.

There will be a briefing session in Michaelmas term.

Syllabus
Students choose five topics in preferential order from a list to be published in Michaelmas term. They will be assigned to three topics out of their list. Students are assessed on one of these topics which may not necessarily be their first choice topic.

The topics to be offered in 2024-25 are yet to be decided but to give an indicative idea of the types of topics, the ones offered in 2023-24 were:

Imitation learning Prof A. Vlachos
The Future of Large Language Models: data architectures ethics Dr M. Tomalin
Physics and Geometry in Machine Learning Dr C. Mishra
Diffusion Models and SDEs Francisco Vargas, Dr C. H. Ek
Explainable Artificial Intelligence M. Espinosa, Z. Shams, Prof M. Jamnik
Unconventional approaches to AI Dr S. Banerjee
Narratives in Artificial Intelligence and Machine Learning Prof N. Lawrence
Multimodal Machine Learning K. Hemker, N. Simidjievski, Prof M. Jamnik
Deep Reinforcement Learning S. Morad, Dr C. H. Ek
Automation in Proof Assistants A. Jiang, W. Li, Prof M. Jamnik
On completion of this module, students should:

be in a strong position to contribute to the research topics covered;
understand the fundamental methods (algorithms, data analysis, specific tasks) underlying each topic;
and be familiar with recent research papers and advances in the field.
Coursework
Students will typically work in groups to give a presentation on assigned papers. Alternatively, a topic may include practical sessions. Each topic will typically consist of one preliminary lecture

followed by 3 reading and discussion sessions, or several lectures followed by a practical session. A typical topic can accommodate up to 9 students presenting papers. There will be at least 10 minutes general discussion per session.

Full coursework details will be published by October.

Assessment
Coursework will be marked by the topic leaders and second marked by the module conveners.

Participation in all assigned topics, 10%
Presentation or practical work or similar (for one of the chosen topics), 20%
Topic coursework (for one of the chosen topics), 70%
Individual topic coursework will be published late Michaelmas term.

Assessment criteria for topic coursework will follow project assessment criteria here:
https://www.cl.cam.ac.uk/teaching/exams/acs_project_marking.pdf

Please note that students will be assessed on one of their three chosen topics but this may not be their first choice.

Recommended reading
To be confirmed by each topic convenor.

COMPUTING FOR COLLECTIVE INTELLIGENCE
Prerequisites: Familiarity with core machine learning paradigms - Basic calculus (analytical skills) - Basic discretre optimization (combinatorics)
Moodle, timetable

Objectives
There is a substantial body of academic work demonstrating that complex life is built by cooperation across scales: collections of genes cooperate to produce organisms, cells cooperate to produce multi-cellular organisms and multicellular animals cooperate to form complex social groups. Arguably, all intelligence is collective intelligence. Yet, to-date, many artificially intelligent agents (both embodied and virtual), are generally not conceived from the ground up to interact with other intelligent agents (be it machines or humans). The canonical AI problem is that of a monolithic and solitary machine confronting a non-social environment. This course aims to balance this trend by (i) equipping students with conceptual and practical knowledge on collective intelligence from a computational standpoint, and (ii) by conveying various computational paradigms by which collective intelligence can be modelled as well as synthesized.

Assessment
The assessment will be based on a reading-group paper presentation (individual or in pairs), accompanied by a 2-page summary, and a technical position paper (individual work) handed in after the course. The position paper will be assessed based on its technical correctness, strength or arguments,
and clarity of research vision, and will be accompanied by a brief 5-minute pre-recorded talk that summarizes key arguments (any slides used are also submitted as part of the project).

Paper presentation and 1-page summary: 25%
Technical position paper: 75% (4000 word limit)
Recommended reading
Swarm Intelligence : From Natural to Artificial Systems, Bonabeau et al (1999)
Joined-Up Thinking, Hannah Critchlow, (2024)
Supercooperators, Martin Nowak (2011)
A Course on Cooperative Game Theory, Chakravarty et al., (2015)
Governing the Commons: The Evolution of Institutions for Collective Action. Ostrom (1990).

CRYPTOGRAPHY AND PROTOCOL ENGINEERING

Aims

We all use cryptographic protocols every day: whenever we access an https:// website or send a message via WhatsApp or Signal, for example. In this module, students will get hands-on experience of how those protocols are implemented. Students start by writing their own secure messaging protocol from scratch, and then move on to more advanced examples of cryptographic protocols for security and privacy, such as private information retrieval (searching for data without revealing what you're searching for).

Syllabus

This is a practical course in which the first half of each of the 2-hour weekly sessions is a lecture that introduces a topic, while the second half is lab time during which the students can work on their implementations, discuss the topics in small groups, and get help from the lecturers. The module is structured into three blocks as follows:

Cryptographic primitives (2 weeks). Using common building blocks such as symmetric ciphers and hash functions. Implementing selected aspects of elliptic curve Diffie-Hellman and digital signatures. Protocol security, Dolev-Yao model, side-channel attacks.
Secure communication (3 weeks). Authenticated key exchange (e.g. SIGMA, TLS), password-authenticated key exchange (e.g. SPAKE2), forward secrecy, post-compromise security, double ratchet, the Signal protocol.
Private database lookups (3 weeks). Lattice-based post-quantum cryptography, learning with errors, homomorphic encryption, private information retrieval.
In each block, students will be given a description of the algorithms and protocols covered (e.g. a research paper or an RFC standards document), and a code template that the students should use for their implementation. The implementation language will probably be Python (to be confirmed after the practical materials have been developed). Students will also be given a set of test cases to check their code, and will have the opportunity to test their implementation communicating with other students' implementations. Finally, for each block, students will write and submit a lab report explaining their approach and the findings from their implementation.

Objectives

On completion of this module, students should:

Understand that cryptography is not magic! The mathematics can look intimidating, but this module will show students that they needn't be afraid of cryptography.
Gain appreciation for the complexity and careful implementation of real-world cryptography libraries ,and understand why one should prefer vetted implementations.
Be familiar with the mathematical notation and concepts commonly used in descriptions of cryptographic protocols, and be able to understand and implement research papers and RFCs in this field.
Be comfortable with cryptographic primitives commonly used in protocols, and able to correctly use software libraries that implement them.

Have gained hands-on experience of attacks on cryptographic protocols, and an appreciation for the difficulty of making these protocols correct.
Have been exposed to ideas and techniques from recent research, which may be useful for their own future research.
Have practised technical writing through lab reports.
Assessment
Students will be provided with code skeletons in one or two different programming languages (probably Python, maybe Rust) to avoid them struggling with setup issues. For each assignment, students are required to produce a working implementation of the protocols discussed in the course, using the programming language and skeleton provided. Here, "working" means that the algorithms are functionally correct, but they are not production-quality (in particular, no side-channel countermeasures such as constant-time algorithms are required). For some primitives (e.g. symmetric ciphers and hash functions) existing libraries should be used, whereas other cryptographic algorithms will be implemented from scratch. Students should submit their code as a Docker container that can be run against specified test cases.

After each of the three blocks, students will be asked to submit a lab report of approximately 2,000 words along with their code for that block. In the lab report, the students should explain how their implementation works and why it is correct, as well as any findings from the work (e.g. any limitations or trade-offs they found with the protocols). The report structure and marking scheme will be specified in advance. The lab reports provide an opportunity for students to provide critical insights and creativity. For instance, they might compare their implementation with existing real-world implementations which provide additional features and guarantees.

Each of the three lab reports (along with the associated code) will be marked, forming the basis of assessment, and feedback on the reports will be given to the students before the next submission is due, so that students can take the feedback on board for their next submission. Of the three reports, the worst mark will be discarded, and the other two reports each contribute 50% of the final mark. As such, students might decide to submit only two reports.

Students may discuss their work with others, but the implementations and lab reports must be individual work.

Recommended reading
Textbook: Jonathan Katz and Yehuda Lindell. Introduction to Modern Cryptography (3rd edition). CRC Press, 2020.

The textbook covers prerequisite knowledge on cryptographic primitives, such as ciphers, MACs, hash functions, and signatures. This book is also used in the Part II Cryptography course. For the protocols we cover in this course, we are not aware of a suitable textbook; instead we will use research papers, lecture slides, and RFCs as resources, which will be provided at the start of the module.

DISTRIBUTED LEDGER TECHNOLOGIES: FOUNDATIONS AND APPLICATIONS

Aims
This reading group course examines foundations and current research into distributed ledger (blockchain) technologies and their applications. Students will read, review, and present seminal research papers in this area. Once completed, students should be able to integrate blockchain technologies into their own research and gain familiarity with a range of research skills.

Lectures
Introduction
Consensus protocols
Bitcoin and its variants
Ethereum, smart contracts, and other permissionless DLTs
Hybrid and permissioned DLTs
Applications
Learning objectives
There are two broad objectives: to acquire familiarity with a body of work in the area of distributed ledgers and to learn some specific research skills:

How to read a paper
How to review a paper
How to analyze a paper's strengths and weaknesses
Written and oral presentation skills
Assessment
You are expected to read all assigned papers and submit paper reviews each week. Each review must either follow the provided review form [PDF] [Latex source]. Each "review" is worth 5% of your total mark, and is marked out of 100 with 60 a passing grade. Marks will be awarded and penalties for late submission applied according to ACS Assessment Guidelines.

One paper review for the first week, then two paper reviews each week for 6 weeks (13 reviews, 5% each) 65% (approx 600 words per review)
Summative essay 25% (max 3000 words)
Presentation 5%
100% attendance in class 5% (2 marks deducted per missed class)
Recommended Reading
Narayanan, A. , Bonneau, J., Felten, E., Miller, A. and Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.
(2016 Draft available here:
https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf)

EXPLAINABLE ARTIFICIAL INTELLIGENCE

Prerequisites: A solid background in statistics, calculus and linear algebra. We strongly recommend some experience with machine learning and deep neural networks (to the level of the first chapters of Goodfellow et al.'s "Deep Learning"). Students are expected to be comfortable reading and writing Python code for the module's practical sessions.
Moodle, timetable

Aims

The recent palpable introduction of Artificial Intelligence (AI) models to everyday consumer-facing products, services, and tools brings forth several new technical challenges and ethical considerations. Amongst these is the fact that most of these models are driven by Deep Neural Networks (DNNs), models that, although extremely expressive and useful, are notoriously complex and opaque. This "black-box" nature of DNNs limits their ability to be successfully deployed in critical scenarios such as those in healthcare and law. Explainable Artificial Intelligence (XAI) is a fast-moving subfield of AI that aims to circumvent this crucial limitation of DNNs by either

(i) constructing human-understandable explanations for their predictions,

or (ii) designing novel neural architectures that are interpretable by construction.

In this module, we will introduce the key ideas behind XAI methods and discuss some of the important application areas of these methods (e.g., healthcare, scientific discovery, debugging, model auditing, etc.). We will approach this by focusing on the nature of what constitutes an explanation, and discussing different ways in which explanations can be constructed or learnt to be generated as a by-product of a model. The main aim of this module is to introduce students to several commonly used approaches in XAI, both theoretically in lectures and through hands-on exercises in practicals, while also bringing recent promising directions within this field to their attention. We hope that, by the end of this module, students will be able to directly contribute to XAI research and will understand how methods discussed in this module may be powerful tools for their own work and research.

Syllabus

Overview and taxonomy of XAI (why is explainability needed, definition of terms, taxonomy of the XAI space, etc.)
Perturbation-based feature attribution (e.g., LIME, Anchors, SHAP, etc.)
Propagation-based feature attribution methods (e.g., Relevance Propagation, Saliency methods, etc.)
Concept-based explainability (Net2Vec, T-CAV, ACE, etc.)
Interpretable architectures (CBMs and variants, Concept Whitening, SENNs, etc.)
Neurosymbolic methods (DeepProbLog, Neural Reasoners, etc.)
Sample-based Explanations (Influence functions, ProtoPNets, etc.)
Counterfactual explanations
Proposed Schedule

The 16 hours of lectures across 8 weeks will be divided as follows:
Week 1: 1h lecture + 1h lecture
Week 2: 1h lecture + 1h reading group & presentations
Week 3: 1h lecture + 1h reading group & presentations
Week 4: 2h practical
Week 5: 1h lecture + 1h reading group & presentations
Week 6: 2h practical
Week 7: 1h lecture + 1h reading group & presentations
Week 8: 1h reading group & presentation + 1h reading group & presentations

In weeks where lectures are planned, one-hour lecture slots will intercalate with one hour of student paper presentations. During each paper presentation session, three students will present a paper related to the topic covered in the earlier lecture that week for about 10 minutes each + 5 minutes of questions. At the end of all paper presentations, there will be a discussion on all the papers. The order of student presentations will be allocated randomly during the first week so that students know in advance when they are expected to present. For the sake of fairness, we will release the paper to be presented by each student a week before their presentation slot.

Objectives
By the end of this module, students should be able to:

Recognise and identify key concepts in XAI together with their connection to related subfields in AI such as fairness, accountability, and trustworthy AI.
Understand how to use, design, and deploy model-agnostic perturbation methods such as LIME, Anchors, and RISE. In particular, students should understand the connection between feature importance and cooperative game theory, and its uses in methods such as SHAP.
Identify the uses and limitations of propagation-based feature importance methods such as Saliency, SmoothGrad, GradCAM, and Integrated Gradients. Students should be able to implement each of these methods on their own and connect the theoretical ideas behind them to practical code, exploiting modern frameworks' auto-differentiation.
Understand what concept learning is and what limitations it overcomes compared to traditional feature-based methods. Specifically, students should understand how probing a DNN's latent space may be exploited for learning useful concepts for explainability.
Reason about the key components of inherently interpretable architectures and neuro-symbolic methods and understand how interpretable neural networks can be designed from first principles.
Elaborate on what sample-based explanations are and how influence functions and prototypical architectures such as ProtoPNet can be used to construct such explanations.
Explain what counterfactual explanations are, how they are related to causality, and under which conditions they may be useful.
Upon completion of this module, students will have the technical background and tools to use XAI as part of their own research or partake in XAI research itself. Moreover, we hope that by detailing a clear timeline of how this relatively young subfield has developed, students may be

able to better understand what are some fundamental open questions in this area and what are some promising directions that are currently actively being explored.

Assessment
(10%) student presentation: each student will be randomly assigned a presentation slot at the beginning of the course. We will then distribute a paper for them to present in their slot a week before the presentation's scheduled time. Students are expected to prepare a 10-minute presentation of their assigned paper where they will present the motivation of their assigned work and discuss the main methodology and findings reported in that paper. We will encourage students to focus on fully communicating the intuition of the work in their assigned papers and try and connect it with ideas that we have previously discussed in previous lectures. All students not presenting each week will be asked to submit one question pertaining to each paper presented that week before the paper presentations.

(20%) practical exercises: We will run two practical sessions where students will be asked to perform a series of exercises that require them to use concepts we have introduced in lecture up to that point. For each practical session, we will prepare a colab notebook to guide the student through exercises and we will ask the students to submit their solutions through this colab notebook. We expect students to complete about ⅔ of the exercises in the practical class and complete the rest at home as homework. Each practical will be worth 10%.

(70%) mini-project: At the end of week 3, we will hand out a list of papers for students to select their mini-projects from. Each mini-project will consist of a student selecting a paper from our list and reimplementing and expanding the key idea in the paper. We encourage students to be as creative as they want with how they drive their mini-project once the paper has been selected. For example, they can reimplement the technique in the paper and combine it with methodologies from other works we discussed in lecture, or they can apply their paper's methodology to a new domain, datasets, or setup, where the technique may offer interesting and potentially novel insights. We will ask all students to submit a report in a workshop format of up to 4,000 words. This report, due roughly a week after Lent term ends, should describe their methodology, experiments, and results. A crucial aspect of this report involves explaining the rationale behind different choices in methodology and experiments, as well as elaborating on the choices made and hypotheses tested throughout their mini-project (potentially showing a deep understanding of the work they are basing their mini-project on). To aid students with selecting their projects and making progress on them, we will hold regular office hours when students can come to discuss their progress and questions with us.

Recommended reading
Textbooks

* Christoph Molnar, "Interpretable Machine Learning". (2022):
https://christophm.github.io/interpretable-ml-book/

Online Courses and Tutorial

* Su-in Lee and Ian Cover, “CSEP 590B Explainable AI” University of Washington* Explaining Machine Learning Predictions: State-of-the-art, Challenges, and Opportunities

* On Explainable AI: From Theory to Motivation, Industrial Applications, XAI Coding & Engineering Practices - AAAI 2022 Turorial

Survey papers

* Arrieta, Alejandro Barredo, et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI." Information fusion 58 (2020): 82-115.

* Rudin, Cynthia, et al. "Interpretable machine learning: Fundamental principles and 10 grand challenges." Statistics Surveys 16 (2022): 1-85.

FEDERATED LEARNING: THEORY AND PRACTICE
Prerequisites: It is strongly recommended that students have previously (and successfully) completed an undergraduate machine learning course - or have equivalent experience through open-source material (e.g., lectures or similar). An example course would be: Part1B "Artificial Intelligence". Students should feel comfortable with SGD and optimization methods used to train current popular neural networks and simple forms of neural network architectures.
Moodle, timetable

Objectives
This course aims to extend the machine learning knowledge available to students in Part I (or present in typical undergraduate degrees in other universities), and allow them to understand how these concepts can manifest in a decentralized setting. The course will consider both theoretical (e.g., decentralized optimization) and practical (e.g., networking efficiency) aspects that combine to define this growing area of machine learning.

At the end of the course students should:

Understand popular methods used in federated learning
Be able to construct and scale a simple federated system
Have gained an appreciation of the core limitations to existing methods, and the approaches available to cope with these issues
Developed an intuition for related technologies like differential privacy and secure aggregation, and are able to use them within typical federated settings
Can reason about the privacy and security issues with federated systems
Lectures
Course Overview. Introduction to Federated Learning.
Decentralized Optimization.
Statistical and Systems Heterogeneity.
Variations of Federated Aggregation.
Secure Aggregation.
Differential Privacy within Federated Systems.
Extensions to Federated Analytics.
Applications to Speech, Video, Images and Robotics.
Lab sessions
Federating a Centralized ML Classifier.
Behaviour under Heterogeneity.
Scaling a Federated Implementation.
Exploring Privacy with Federated Settings
Recommended Reading
Readings will be assigned for each lecture. Readings will be taken from either research papers, tutorials, source code or blogs that provide more comprehensive treatment of taught concepts.

Assessment - Part II Students

Four labs are performed during the course, and students receive 10% of their total grade for work done as part of each lab. (For a total of 40% of the total grade from lab work alone). Labs will primarily provide hands-on teaching opportunities, that are then utilized within the lab assignment which is completed outside of the lab contact time. MPhil and Part III students will be given additional questions to answer within their version of the lab assignment which will differ from the assignment given to Part II CST students.

The remainder of the course grade (60%) will be given based on a hands-on project that applies the concepts taught in lectures and labs. This hands-on project will be assessed based on upon a combination of source code, related documentation and brief 8-minute pre-recorded talk that summarizes key project elements (any slides used are also submitted as part of the project). Please note, that in the case of Part II CST students, the talk itself is not examinable -- as such will be made optional to those students.

A range of possible practical projects will be described and offered to students to select from, or alternatively students may propose their own. MPhil and Part III students will select from a project pool that is separate from those offered to Part II CST students. MPhil and Part III projects will contain a greater emphasis on a research element, and the pre-recorded talks provided by this student group will focus on this research contribution. The project will be assessed on the level of student understanding demonstrated, the degree of difficulty, correctness of implementation -- and for Part III/MPhil students the additional criteria of the quality and execution of the research methodology, and depth and quality of results analysis.

This project can be done individually or in groups -- although individual projects will be strongly encouraged. It will be required the project is performed using a code repository that also will contain all documentation -- access to this repository will be shared with course staff (e.g., lecturer and TAs). Where needed, marks assigned to students within a group will be differentiated using this repository as an input. Furthermore if groups are formed, members must be either entirely from Part III/MPhil students or Part II CST, i.e., these two student groups should not mix to form a project group.

Projects will be made available publicly. A maximum word count for written contributions for the project will be enforced.

Assessment - MPhil / Part III Students
50% final mini-project. This hands-on project will be assessed based on upon a combination of source code, related documentation and brief 8-minute pre-recorded talk that summarizes key project elements (any slides used are also submitted as part of the project).

30% midterm lab report (a written report of a bit more depth of thought than required in a lab, the report provides the results of experiments -- with the skills to perform these experiments coming from lab sessions), 20% for two individual hands-on labs. Labs will primarily provide hands-on teaching opportunities, that are then utilized within the lab assignment which is completed outside of the lab contact time. There will also be additional questions to answer within their version of the lab assignment.

For the mini-project, a range of possible practical projects will be described and offered to students to select from, or alternatively students may propose their own. The project will be assessed on the level of student understanding demonstrated, the degree of difficulty, correctness of implementation, the quality and execution of the research methodology, and depth and quality of results analysis.

The project can be done individually or in groups -- although individual projects will be strongly encouraged. It will be required the project is performed using a code repository that also will contain all documentation -- access to this repository will be shared with course staff (e.g., lecturer and TAs). Where needed, marks assigned to students within a group will be differentiated using this repository as an input. Furthermore if groups are formed, members must be either entirely from Part III or entirely from MPhil students, i.e., these two student groups should not mix to form a project group.

Projects will be made available publicly. A maximum word count for written contributions for the project will be enforced.

Advanced Material sessions - MPhil / Part III Students
These sessions are mandatory for the MPhil and Part III students. Part II CST students may attend if they wish

Topics and announced the first week of class. Selected to extend beyond topics covered in lectures and labs. Content is a mixture of sessions run in the lecture room the are either (1) presentations by guest lectures by an invited domain expert or (2) class-wide discussions regarding one or more related academic papers. Paper discussions will require students to read the paper ahead of the lecture, and a brief discussion primer presentation will be given students before discussions begin.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

GEOMETRIC DEEP LEARNING

Prerequisites: Experience with machine learning and deep neural networks is recommended. Knowledge of concepts from graph theory and group theory is useful, although the relevant parts will be explicitly retaught.
Moodle, timetable

Aims and Objectives
Most of the patterns we see in nature can be elegantly reasoned about using spatial symmetries—transformations that leave underlying objects unchanged. This observation has had direct implications for the development of modern deep learning architectures that are seemingly able to escape the “curse of dimensionality” and fit complex high-dimensional tasks from noisy real-world data. Beyond this, one of the most generic symmetries—the permutation—will prove remarkably powerful in building models that reason over graph structured-data, which is an excellent abstraction to reason about naturally-occurring, irregularly-structured data. Prominent examples include molecules (represented as graphs of atoms and bonds, with three-dimensional coordinates provided), social networks and transportation networks. Several already-impacted application areas include traffic forecasting, drug discovery, social network analysis and recommender systems. The module will provide the students the capability to analyse irregularly- and nontrivially-structured data in an effective way, and position geometric deep learning in a proper context with related fields. The main aim of the course is to enable students to make direct contributions to the field, thoroughly assimilate the key concepts in the area, and draw relevant connections to various other fields (such as NLP, Fourier Analysis and Probabilistic Graphical Models). We assume only a basic background in machine learning with deep neural networks.

Learning outcomes
The framework of geometric deep learning, and its key building blocks: symmetries, representations, invariance and equivariance
Fundamentals of processing data on graphs, as well as impactful application areas for graph representation learning
Theoretical principles of graph machine learning: permutation invariance and equivariance
The three "flavours" of spatial graph neural networks (GNNs) (convolutional, attentional, message passing) and their relative merits. The Transformer architecture as a special case.
Attaching symmetries to graphs: CNNs on images, spheres and manifolds, Geometric Graphs and E(n)-equivariant GNNs
Relevant connections of geometric deep learning to various other fields (such as NLP, Fourier Analysis and Probabilistic Graphical Models)
Lectures
The lectures will cover the following topics:

Learning with invariances and symmetries: geometric deep learning. Foundations of group theory and representation theory.

Why study data on graphs? Success stories: drug screening, travel time estimation, recommender systems. Fundamentals of graph data processing: network science, spectral clustering, node embeddings.
Permutation invariance and equivariance on sets and graphs. The principal tasks of node, edge and graph classification. Neural networks for point clouds: Deep Sets, PointNet; universal approximation properties.
The three flavours of spatial GNNs: convolutional, attentional, message passing. Prominent examples: GCN, SGC, ChebyNets, MoNet, GAT, GATv2, IN, MPNN, GraphNets. Tradeoffs of using different GNN variants.
Graph Rewiring: how to apply GNNs when there is no graph? Links to natural language processing---Transformers as a special case of attentional GNNs. Representative methodologies for graph rewiring: GDC, SDRF, EGP, DGCNN.
Expressive power of graph neural networks: the Weisfeiler-Lehman hierarchy. GINs as a maximally expressive GNN. Links between GNNs and graph algorithms: neural algorithmic reasoning.
Combining spatial symmetries with GNNs: E(n)-equivariant GNNs, TFNs, SE(3)-Transformer. A deep dive into AlphaFold 2.
Worked examples: Circulant matrices on grids, the discrete Fourier transform, and convolutional networks on spheres. Graph Fourier transform and the Laplacian eigenbasis.
Practicals
The practical is designed to complement the knowledge learnt in lectures and teach students to derive additional important results and architectures not directly shown in lectures. The practical will be given as a series of individual exercises (each either code implementation or proof/derivation). Each of these exercises can be individually assessed based on a specified mark budget.

Possible practical topics include the study of higher-order GNNs and equivariant message passing.

Assessment
(60%) Group Mini-project (writeup) at the end of the course. The mini projects can either be self-proposed, or the students can express their preference for one of the provided topics, the list of which will be announced at the start of term. The projects will consist of implementing and/or extending graph representation learning models in the literature, applying them to publicly available datasets. Students will undertake the project in pairs and submit a joint writeup limited to 4,000 words (in line with other modules); appendix of work logs to be included but ungraded;

(10%) Short presentation and viva: students will give a short presentation to explain their individual contribution to the mini-project and there will be a short viva following.

(30%) Practical work completion. Completing the exercises specified in the practical to a satisfactory standard. The practical assessor should be satisfied that the student derived their answers using insight gained from the course; coupled with original thought, not by simple

copy-pasting of relevant related work. The students would submit code and a short report, which would then be marked in line with the predetermined mark budget for each practical item.
The students will learn how to run advanced architectures on GPU but no specific need for dedicated GPU resources. Practicals will be made possible to do on CPU; if required, students can use GPUs on publicly available free services (such as Colab) for their mini-project work.

References
The course will be based on the following literature:

"Geometric Deep Learning: Grids, Graphs, Groups, Geodesics, and Gauges", by Michael Bronstein, Joan Bruna, Taco Cohen and Petar Veličković
"Graph Representation Learning", by Will Hamilton
"Deep Learning", by Ian Goodfellow, Yoshua Bengio and Aaron Courville.

MOBILE HEALTH

Aims

The course aims to explore how wearable and mobile systems sensors can be used to gather data relevant to understand health, how the data can be analysed with advanced signal processing and machine learning and the performance of these systems in terms of diagnostics and disease progression detection.

Syllabus

Course Overview. Introduction to Mobile Health. Evaluation metrics and methodology. Basics of Signal Processing.
Inertial Measurement Units, Human Activity Recognition (HAR) and Gait Analysis and Machine Learning for IMU data.
Radios, Bluetooth, GPS and Cellular. Epidemiology and contact tracing, Social interaction sensing and applications. Location tracking in health monitoring and public health.
Audio Signal Processing. Voice and Speech Analysis: concepts and data analysis. Body Sounds analysis.
Photoplethysmogram and Light sensing for health (heart and sleep)
Contactless and wireless behaviour and physiological monitoring
Generative Models for Wearable Health Data
Topical Guest Lectures

Objectives

The course aims to explore how wearable and mobile systems sensors can be used to gather data relevant to understand health, how the data can be analysed with advanced signal processing and machine learning and the performance of these systems in terms of diagnostics and disease progression detection.

Roughly, each lecture contains a theory part about the working of “sensor signals” or “data analysis methods” and an application part which contextualises the concepts.

At the end of the course students should: Understand how mobile/wearable sensors capture data and their working. Understand different approaches to acquiring and analysing sensor data from different types of sensors. Understand the concept of signal processing applied to time series data and their practical application in health. Be able to extract sensor data and analyse it with basic signal processing and machine learning techniques. Be aware of the different health applications of the various sensor techniques. The course will also touch on privacy and ethics implications of the approaches developed in an orthogonal fashion.

Recommended Reading

Please see Course Materials for recommended reading for each session.

Assessment - Part II students

Two assignments will be based on two datasets which will be provided to the students:

Assignment 1 (shared for Part II and Part III/MPhil): this will be based on a dataset and will be worth 40% of the final mark. The task of the assessment will be to perform pre-processing and basic data analysis in a "colab" and an answer sheet of no more than 1000 words.

Assignment 2 (Part II): This assignment (worth 60% of the final mark) will be a fuller analysis of a dataset focusing on machine learning algorithms and metrics. Discussion and interpretation of the findings will be reported in a colab and a report of no more than 1200 words.

Assessment - Part III and MPhil students
Two assignments will be based on datasets which will be provided to the students:

Assignment 1: this will be based on a dataset and will be worth 40% of the final mark. The task of the assessment will be to perform pre-processing and basic data analysis in a "colab" and an answer sheet of no more than 1000 words.

Assignment 2: The second assignment (worth 60% of the final mark) will be based on multiple datasets. The students will be asked to compare and contract ML algorithms/solutions (trained and tested on the different data) and discuss the findings and interpretation in terms of health context. This will be in the form of a colab and a report of 1800 words.

Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

MULTICORE SEMANTICS AND PROGRAMMING
Prerequisites: Some familiarity with discrete mathematics (sets, partial orders, etc.) and with sequential Java programming will be assumed. Experience with operational semantics and with some concurrent programming would be helpful.
Moodle, timetable

Aims
In recent years multiprocessors have become ubiquitous, but building reliable concurrent systems with good performance remains very challenging. The aim of this module is to introduce some of the theory and the practice of concurrent programming, from hardware memory models and the design of high-level programming languages to the correctness and performance properties of concurrent algorithms.

Lectures
Part 1: Introduction and relaxed-memory concurrency [Professor P. Sewell]

Introduction. Sequential consistency, atomicity, basic concurrent problems. [1 block]
Concurrency on real multiprocessors: the relaxed memory model(s) for x86, ARM, and IBM Power, and theoretical tools for reasoning about x86-TSO programs. [2 blocks]
High-level languages. An introduction to C/C++11 and Java shared-memory concurrency. [1 block]
Part 2: Concurrent algorithms [Dr T. Harris]

Concurrent programming. Simple algorithms (readers/writers, stacks, queues) and correctness criteria (linearisability and progress properties). Advanced synchronisation patterns (e.g. some of the following: optimistic and lazy list algorithms, hash tables, double-checked locking, RCU, hazard pointers), with discussion of performance and on the interaction between algorithm design and the underlying relaxed memory models. [3 blocks]
Research topics, likely to include one hour on transactional memory and one guest lecture. [1 block]
Objectives
By the end of the course students should:

have a good understanding of the semantics of concurrent programs, both at the multprocessor level and the C/Java programming language level;
have a good understanding of some key concurrent algorithms, with practical experience.
Assessment - Part II Students
Two assignments each worth 50%

Recommended reading
Herlihy, M. and Shavit, N. (2008). The art of multiprocessor programming. Morgan Kaufmann.

Coursework
Coursework will consist of assessed exercises.

Practical work
Part 2 of the course will include a practical exercise sheet. The practical exercises involve building concurrent data structures and measuring their performance. The work can be completed in C, Java, or similar languages.

Assessment - Part III and MPhil Students
Assignment 1, for 50% of the overall grade. Written coursework comprising a series of questions on hardware and software relaxed-memory concurrency semantics.
Assignment 2, for 50% of the overall grade. Written coursework comprising a series of questions on the design of mutual exclusion locks and shared-memory data structures.
Further Information
Current Cambridge undergraduate students who are continuing onto Part III or the MPhil in Advanced Computer Science may only take this module if they did NOT take it as a Unit of Assessment in Part II.

This module is shared with Part II of the Computer Science Tripos. Assessment will be adjusted for the two groups of students to be at an appropriate level for whichever course the student is enrolled on. Further information about assessment and practicals will follow at the first lecture.

REINFORCEMENT LEARNING

Aims

The aim of this module is to introduce students to the foundations of the field of reinforcement learning (RL), discuss state-of-the-art RL methods, incentivise students to produce critical analysis of recent methods, and prompt students to propose novel solutions that address shortcomings of existing methods. If judged by the headlines, RL has seen an unprecedented success in recent years. However, the vast majority of RL methods still have shortcomings that might not be apparent at first glance. The aim of the course is to inspire students by communicating the promising aspects of RL, but ensure that students develop the ability to produce a critical analysis of current RL limitations.

Objectives

Students will learn the following technical concepts:

fundamental RL terminology and mathematical formalism; a brief history of RL and its connection to neuroscience and biological systems
RL methods for discrete action spaces, e.g. deep Q-learning and large-scale Monte Carlo Tree Search
methods for exploration, modelling uncertainty, and partial observability for RL
modern policy gradient and actor-critic methods
concepts needed to construct model-based RL and Model Predictive Control methods
approaches to make RL data-efficient and ways to enable simulation-to-reality transfer
examples of fine-tuning foundation models and large language models (LLMs) with human feedback; safe RL concepts; examples of using RL for safety validation
examples of using RL for scientific discovery
Students will also gain experience with analysing RL methods to uncover their strengths and shortcomings, as well as proposing extensions to improve performance.

Finally, students will gain skills needed to create and deliver a successful presentation in a format similar to that of conference presentations.

With all of the above, students who take part in this module will be well-prepared to start conducting research in the field of reinforcement learning.

Syllabus

Topic 1: Introduction and Fundamentals

Overview of RL: foundational ideas, history, and books; connection to neuroscience and biological systems, recent industrial applications and research demonstrations
Mathematical fundamentals: Markov decision processes, Bellman equations, policy and value iteration, temporal difference learning
Topic 2: RL in Discrete Action Spaces

Q-learning, function approximation and deep Q-learning; nonstationarity in RL and its implications for deep learning; example applications (video games; initial example: Atari)
Monte Carlo Tree Search; example applications (AlphaGo)
Topic 3: Exploration, Uncertainty, Partial Observability

Multi-armed bandits, Bayesian optimisation, regret analysis
Partially observable Markov decision process; belief, memory, and sequence modelling (probabilistic methods, recurrent networks, transformers)
Topic 4: Policy Gradient and Actor-critic Methods for Continuous Action Spaces

Importance sampling, policy gradient theorem, actor-critic methods (SPG, DDPG)
Proximal policy optimisation; example applications
Topic 5: Model-based RL and Model Predictive Control

Learning dynamics models (graph networks, stochastic processes, diffusion models, physics-based models, ensembles); planning with learned models
Model predictive control; example applications (real-time control)
Topic 6: Data-efficient RL and Simulation-to-reality Transfer

Data-efficient learning with probabilistic methods from real data (e.g. policy search in robotics), real-to-sim inference and differentiable simulation, data-efficient simulation-to-reality transfer
RL for physical systems (successful examples in locomotion, open problems in contact-rich manipulation, applications to logistics, energy, and transport systems); examples of RL for healthcare.
Topic 7: RL with Human Feedback ; Safe RL and RL for Validation

Fine-tuning large language models (LLMs) and other foundation models with human feedback (TRLX,RL4LMs, a light-weight overview of RLHF)
A review of SafeRL, example: optimising commercial HVAC systems using policy improvement with constraints; improving safety using RL for validation: examples in autonomous driving and autonomous flying and aircraft collision avoidance
Topic 8: RL for Scientific Discovery; Student Presentations

Examples of RL for molecular design and drug discovery, active learning for synthesising new materials, RL for nuclear fusion experiments
Student presentations (based on essays and mini-projects) for other topics in RL, e.g. multi-agent RL, hierarchical RL, RL for hyperparameter optimisation and NN architecture search, RL for multi-task transfer, lifelong RL, RL in biological systems, etc.
Assessment
The assessment for this module consists of:

Essay and mini-project (60%)

Students will choose a concrete RL method from the literature, then complete a 2-part essay described below:
Essay Part 1 (1500 words, 20%): Introduce a formal description of the method and explain how the method extended the state-of-the-art at the time of its publication
Essay Part 2 or a mini-project (2500 words, 40%): Provide a critical analysis of the chosen RL method.
Presentation of the essay and mini-project results (15%)
students will be expected to create and record a 15-minute video presentation of the analysis described in their essay and mini-project.
Short test of RL theory fundamentals (15%)
to test the understanding of RL fundamentals (~15 minutes, in-class, closed book)
Participation in seminar discussions (10%)
take an active part in the seminars by asking clarifying questions and mentioning related works.
Recommended Reading
Books

[S&B] Reinforcement Learning: An Introduction (second print edition). Richard S. Sutton, Andrew G. Barto. [Available from the book’s website as a free PDF updated in 2022]

[CZ] Algorithms for Reinforcement Learning. Csaba Szepesvari. [Available from the book’s website as a free PDF updated in 2019]

[MK] Algorithms for Decision Making. Mykel J. Kochenderfer Tim A. WheelerKyle H. Wray. [Available from the book’s website as a free PDF updated in 2023]

[DB] Reinforcement Learning and Optimal Control. Dimitri Bertsekas. [Available from the book’s website as a free PDF updated in 2023]

Presentation guidelines

[KN] Ten simple rules for effective presentation slides. Kristen M.Naegle. PLoS computational biology 17, no. 12 (2021).

THEORIES OF SOCIO-DIGITAL DESIGN FOR HUMAN-CENTRED AI

Prerequisites: This course is appropriate to any ACS student, and will assist in the development of critical thinking, argument and long-form writing skills. Prior experience of essay-based humanities subjects at university or secondary school will be beneficial, but not required.
Moodle, timetable

Aims

This module is a theoretically-oriented advanced introduction to the broad field of human-computer interaction, extending to consider topics such as intelligent user interfaces, critical and speculative design, participatory design, cognitive models of users, human-centred, as well as more-than-human-centred design, and others. The course will not address purely engineering approaches to the development of user interfaces (unless there is a clear theoretical question being addressed), but allow students to effectively link the political, social, and ethical considerations in HCI to specific technical and conceptual design challenges. The module builds on the Practical Research in Human-Centred AI course offered in Michaelmas (developed with researchers at Microsoft Research Cambridge) and a collaboration with the Leverhulme Centre for the Future of Intelligence at Cambridge. Participants may include visitors from these groups and/or interdisciplinary research students and academic guests from other University departments, including Cambridge Digital Humanities.

Syllabus

The syllabus will remain broadly within the area of human-computer interaction, including theories of design practice and the social contexts of technology use. Individual seminar topics will be selected in response to contemporary and recent research developments, in consultation with members of the class and visiting contributors.

Representative topics in the next year are likely to include:

Responsible AI and HCI
Intersectional design for social justice
Participatory design and co-design
Sustainable AI and more-than-human-centred design
Usefulness and usability of ethical design toolkits
The EU AI Act in design practice
AI and interface design for transparency
Designing otherwise: on anarchist HCI

Objectives

On completion of this module, students should have developed facility in discussing and critiquing the aims of their research, especially for an audience drawn from other academic disciplines, including the following skills:

Theoretical motivation and defence of a research question.

Consideration of a research proposal from one or more alternative theoretical perspectives.
Potential critique of the theoretical basis for a programme of research.
Coursework
In advance of each seminar, all participants must read the essential reading - generally one long, plus one or two short papers. Further reading is suggested for some seminars. Some weeks, instead of shorter papers, students will be asked to explore specific design tools or resources.

Before seminars 2-8, each student will submit a written commentary on the paper, either generated using a large language model (LLM), or written in the style of an LLM, supplemented by a short original observation.

Each member of the class must select one of the 8 seminar themes as the subject for a more extended critical review essay. The critical review should be written as an academic research essay, not in LLM style.

Each seminar will include an informal design exercise to be carried out in small groups. The purpose of these exercises is to reflect on a variety of practical design resources, by applying those resources to concrete case studies.

Throughout the course, students will be expected to keep a "reflective diary", briefly reporting themes that arise in discussion, reflections on the design exercises, and ways in which the readings relate to their own research interests.

Practical work
There is no practical work element in this course.

Assessment
Written critical review of a single discussion text (20%)
Reflective diary submitted at the end of the module (60%)
In advance of the session, each student will submit a written commentary on the paper, either generated using a large language model (LLM), or written in the style of an LLM. A short original observation should be added (20%)
Recommended reading
To be assigned during the module, as discussed above. Most set readings will be available either from the ACM Digital Library, or from institutional repositories of the authors.

Further Information
Students from the MPhil in the Ethics of AI, Data and Algorithms, MSt in AI, and MPhil in Digital Humanities courses also attend the lectures for this module.

THEORY OF DEEP LEARNING

Prerequisites: A strong background in calculus, probability theory and linear algebra, familiarity with differential equations, optimization and information theory is required. Students need to have studied Deep Neural Networks, familiarity with deep learning terminology (e.g. architectures, benchmark problems) as well as deep learning frameworks (pytorch, jax or similar) is assumed
Moodle, timetable

Aims
The objectives of this course is to expose you to one of the most active contemporary research directions within machine learning: the theory of deep learning (DL). While the first wave of modern DL has focussed on empirical breakthroughs and ever more complex techniques, the attention is now shifting to building a solid mathematical understanding of why these techniques work so well in the first place. The purpose of this course is to review this recent progress through a mixture of reading group sessions and invited talks by leading researchers in the topic, and prepare you to embark on a PhD in modern deep learning research. Compared to typical, non-mathematical courses on deep learning, this advanced module will appeal to those who have strong foundations in mathematics and theoretical computer science. In a way, this course is our answer to the question “What should the world’s best computer science students know about deep learning in 2023?”

Learning Outcomes
This course should prepare the best students to start a PhD in the theory and mathematics of deep learning, and to start formulating their own hypotheses in this space. You will be introduced to a range of empirical and mathematical tools developed in recent years for the study of deep learning behaviour, and you will build an awareness of the main open questions and current lines of attack. At the end of the course you will:

be able to explain why classical learning theory is insufficient to describe the phenomenon of generalization in DL
be able to design and interpret empirical studies aimed at understanding generalization
be able to explain the role of overparameterization: be able to use deep linear models as a model to study implicit regularisation of gradient-based learning
be able to state PAC-Bayes and Information-theoretic bounds, and apply them to DL
be able to explain the connection between Gaussian processes and neural networks, and will be able to study learning dynamics in the neural tangent kernel (NTK) regime.
be able to formulate your own hypotheses about DL and choose tools to prove/test them
leverage your deeper theoretical understanding to produce more robust, rigorous and reproducible solutions to practical machine learning problems.
Syllabus
Each week we'll have two to four student-lead presentations about a research paper chosen from a reading list. The reading list loosely follows the weekly breakdown below (but we adapt it each year based as this is an active research area):

Week 1: Introduction to the topic
Week 2: Empirical Studies of Deep Learning Phenomena
Week 3: Interpolation Regime and “Double Descent” Phenomena
Week 4: Implicit Regularization in Deep Linear Models
Week 5: Approximation Theory
Week 6: Networks in the Infinite Width Limit
Week 7: PAC-Bayes and Information Theoretic Bounds for SGD
Week 8: Discussion and Coursework Spotlight Session

Assessment
Students will be assessed on the following basis:

20% for presentation/content contributed to the module: Each student will have an opportunity to present one of the recommended papers during Weeks 1-7 (30 minute slot including Q&A). For the presentation, students should aim to communicate the core ideas behind the paper, and clearly present the results, conclusions, and future directions. Where possible, students are encouraged to comment on how the work itself fits into broader research goals.
10% for active participation (regular attendance and contribution to discussions during the Q&A sessions).
70% for a group project report, with a word limit of 4000. Either (1) an original research proposal/report with a hypothesis, review of related literature, and ideally preliminary findings, or, (2) reproduction and ideally extension of an existing relevant paper.
Coursework reports are marked in line with general ACS guidelines, reports receiving top marks will have have demonstrable research value (contain an original research idea, extension of existing work, or a thorough reproduction effort which is valuable to the research community).
Additionally, some projects will be suggested during the first weeks of the course, although students are encouraged to come up with their own ideas. Students may be required to participate in group projects, with groups of size 2-3 (the class groups will be separated). For any given project, individual contributions would be noted for assessment though a viva component.
Relationship with related modules
This course can be considered as an advanced follow-up to the Part IIB course on Deep Neural Networks. That course introduces some high level concepts that this course significantly expands on.

This module complements L48: Machine Learning in the Physical World and L46: Principles of Machine Learning Systems, which focus on applications and hardware/systems aspects of ML respectively.

Recommended reading
PNAS Colloquium on the Science of Deep Learning
Mathematics of Machine Learning book by Marc Deisenroth, Aldo Faisal and Cheng Soon Ong.
Probabilistic Machine Learning: An Introduction book by Kevin Murphy
Matus Telgarsky's lecture notes on deep learning theory

These are in addition to the papers which will be discussed in the lectures.

UNDERSTANDING NETWORKED-SYSTEMS PERFORMANCE

Prerequisites: A student must possess knowledge comparable to the undergraduate subjects on Unix Tools, C/C++ programming and Computer Networks. Optionally; a student will be advantaged by doing an introduction to Computer Systems Modelling; such as the Part 2 Computer Systems Modelling subject as well as having a background in measurement methodologies such as that of L50: Introduction to networking and systems measurements provided in the ACS/Part III curriculum.
Moodle, timetable

Aims

This is a practical course, actively building and extending software tools to observe the detailed behaviour of transaction-oriented datacenter-like software. Students will observe and understand sources of user-facing tail latency, including that stemming
from resource contention, cross-program interference, bad software locking, and simple design errors.
A 2-hour weekly hybrid, lecture/lab format permits students continuous monitored progress with complexity of tasks building naturally upon the previous weeks learning.

Objectives

Upon successful completion of the course, students will be able to:

Make order-of-magnitude estimates of software, hardware, and I/O speeds
Make valid measurements of actual software, hardware, and I/O speeds
Create observation facilities, including logs and dashboards, as part of a software system design
Create tracing facilities to fully observe the execution of complex software
Time-align traces across multiple computers
Display dense tracing information in meaningful ways
Reason about the sources of real-time and transaction delays, including cross-program resource interference, remote procedure call delays, and software locking surprises
Fix programs based on the above reasoning, making their response times faster and more robust

Syllabus

Measurement
- Week 1 Intro, Measuring CPU time, rdtsc, Measuring memory access times, Measuring disks,
- Week 2 gettimeofday, logs Measuring networks, Remote Procedure Calls, Multi-threads, locks

Observation
- Week 3 RPC, logs, displaying traces, interference
- Week 4 Antagonist programs, Logging, dashboards, profiling.

KUtrace
- Week 5 Kernel patches, hello world, post-processing
- Week 6 KUtrace multi-CPU time display
- Week 7 Client-server KUtrace, with antagonists and interference

Week 8 Other trace mysteries

Assessment
This is a Lab-based module; assessment is based upon reports covering guided laboratory work performed each week. Assessment will be via two submissions:

20% assignment deadline week 4 based upon Lab work from Weeks 1-4; word target 1000, limit 2000
80% assignment deadline week 8 based upon lab work from weeks 5-8; word target 2000, limit 4000
The intent of the first assessment point is to provide rich feedback to students based upon a 20% assignment permitting focussed and improved work to be executedfor the final assignment.

All work in this module is expected to be the effort of the student. Enough equipment is provisioned to allow each student an independent set of apparatus. While classmembers may find sharing operational experience valuable all assessment is based upon a students sole submission based upon. their own experiments and findings.

Reading Material
Core text

Richard L. Sites, Understanding Software Dynamics, Addison-Wesley Professional Computing Series, 2022
Further reading: papers and presentations that you might find interesting. None are required reading – they are well-written and/or informative.

John K. Ousterhout et al., A trace-driven analysis of the UNIX 4.2 BSD file system ACM SIGOPS Operating Systems Review, Vol. 19, No. 5, Dec, 1985
https://dl.acm.org/doi/pdf/10.1145/323627.323631
Luiz Andr´e Barroso, Jimmy Clidaras, Urs H¨olzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition
John L. Hennessy, David A. Patterson, Computer Architecture, A Quantitative Approach 5th Edition
George Varghese, Network Algorithmics. Morgan Kaufmann
Actually keeping datacenter software up and running – how Google runs production systems Site Reliability Engineering Edited by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy free PDF: https://landing.google.com/sre/book.html
How NOT to run datacenters (27 minute video, funny/sad) dotScale 2014 - Robert Kennedy - Life in the Trenches of healthcare.gov https://www.youtube.com/watch?v=GLQyj-kBRdo
An extended (optional) reading list will also be provided.