Quantum Machine Intelligence (2021) 3: 16
https://doi.org/10.1007/s42484-021-00041-1

RESEARCH ARTICLE

A quantum search decoder for natural language processing

Johannes Bausch[1] · **Sathyawageeswar Subramanian**[1] · **Stephen Piddock**[2,3]

Received: 5 November 2020 / Accepted: 9 February 2021 / Published online: 30 April 2021

Abstract

Probabilistic language models, e.g. those based on recurrent neural networks such as long short-term memory models (LSTMs), often face the problem of finding a high probability prediction from a sequence of random variables over a set of tokens. This is commonly addressed using a form of greedy decoding such as beam search, where a limited number of highest-likelihood paths (the beam width) of the decoder are kept, and at the end the maximum-likelihood path is chosen. In this work, we construct a quantum algorithm to find the globally optimal parse (i.e. for infinite beam width) with high constant success probability. When the input to the decoder follows a power law with exponent $k > 0$, our algorithm has runtime $R^{nf(R,k)}$, where R is the alphabet size, n the input length; here $f < 1/2$, and $f \to 0$ exponentially fast with increasing k, hence making our algorithm always more than quadratically faster than its classical counterpart. We further modify our procedure to recover a finite beam width variant, which enables an even stronger empirical speedup while still retaining higher accuracy than possible classically. Finally, we apply this quantum beam search decoder to Mozilla's implementation of Baidu's *DeepSpeech* neural net, which we show to exhibit such a power law word rank frequency.

Keywords Recurrent neural networks · Quantum algorithms · Quantum search · Parsing · Natural language processing · Quantum speedups

1 Introduction

A recurring task in the context of parsing and neural sequence to sequence models—such as machine translation (Ilya et al. 2011; Sutskever et al. 2014), natural language processing (Schmidhuber 2014) and generative models (Graves 2013)—is to find an optimal path of tokens (e.g. words or letters) from a sequential list of probability distributions. Such a distribution can for instance be produced at the output layer of a recurrent neural network, e.g. a long short-term memory (LSTM). The goal is to decode these distributions by scoring all viable output sequences (paths) under some language model, and finding the path with the highest score.

Nowadays, the de facto standard solution is to use a variant of beam search (Steinbiss et al. 1994; Vijayakumar et al. 2016; Wiseman and Rush 2016; Kulikov et al. 2018; Pratap et al. 2020) to traverse the list of all possible output strings. Beam search stores and explores a constant sized list of possible decoded hypotheses at each step, compared to a greedy algorithm that only considers the top element at each step. Beam search thus interpolates between a simple greedy algorithm and best-first search; but just like greedy search, beam search is not guaranteed to find a global optimum. Furthermore, beam search suffers from sensitivity to the predicted sequence length. Improving the algorithm itself (Murray and Chiang 2018; Yang et al. 2018), as well as finding new decoding strategies (Fan et al. 2018; Holtzman et al. 2020), is an ongoing field of research.

A related task is found in transition based parsing of formal languages, such as context-free grammars (Hopcroft et al. 2001; Zhang and Clark 2008; Zhang and features 2011; Zhu et al. 2015; Dyer et al. 2015). In this model, an input string is processed token by token, and a heuristic prediction (which can be based on various types of

✉ Johannes Bausch
jkrb2@cam.ac.uk

Sathyawageeswar Subramanian
ss2310@cam.ac.uk

Stephen Piddock
stephen.piddock@bristol.ac.uk

[1] CQIF, DAMTP, University of Cambridge, Cambridge, CB3 0WA, UK

[2] Heilbronn Institute for Mathematical Research, Bristol, UK

[3] School of Mathematics, University of Bristol, Bristol BS8 1TW, UK

classifiers, such as feed forward networks) is made on how to apply a transition at any one point. As in generative models and decoding tasks, heuristic parsing employs beam search, where a constant sized list of possible parse trees is retained in memory at any point in time, and at the end the hypothesis optimising a suitable objective function is chosen. Improvements of beam search-based parsing strategies are an active field of research (Buckman et al. 2016; Bohnet et al. 2016; Vilares and Gómez-Rodriguez 2018).

In essence, the problem of decoding a probabilistic sequence with a language model—or probabilistically parsing a formal grammar—becomes one of searching for paths in an exponentially growing tree: since at each step or node the list of possible sequence hypotheses branches, with maximum degree equal to the number of predictions for the next tokens. The goal is to find a path through this search space with the highest overall score. Due to runtime and memory constraints, a tradeoff has to be made which limits any guarantees on the performance of the search strategy.

Quantum computing has shown promise as an emerging technology to efficiently solve some instances of difficult computing tasks in fields ranging from optimisation (Gilyén et al. 2019; Montanaro 2020), linear algebra (Harrow et al. 2009; Berry et al. 2017), number theory and pattern matching (Montanaro 2016; 2017), language processing (Aaronson et al. 2019; Wiebe et al. 2019), machine learning (McClean et al. 2016; Bausch 2018; Wang et al. 2019; Li et al. 2019), to quantum simulation (Lloyd 1996; Babbush et al. 2018; Childs and Su 2019). While quantum computers are not yet robust enough to evaluate any of these applications on sample sizes large enough to claim an empirical advantage, a structured search problem such as language decoding is a prime candidate for a quantum speedup.

Although most naïve search problems can be sped up using Grover's search algorithm (or one of its variants, such as fixed point search or oblivious amplitude amplification), finding good applications for quantum algorithms remains challenging, and super-quadratic (i.e. *faster than Grover*) speedups—such as Shor's for prime factorisation (Shor 1999)—are rare. Recently, several exponentially faster algorithms (such as quantum recommender systems (Kerenidis and Prakash 2016), or dense low rank linear algebra (Wossnig et al. 2018)) have been proven to rely on a quantum random access memory model which, if classically available, can yield an exponential speedup without the need for quantum computing (Tang 2019).

In this work, we develop a quantum search decoder for parsing probabilistic token sequences with a super-quadratic speedup as compared to its classical counterpart. The algorithm can be seen as a generalisation of classical beam search, with potentially infinite beam width; for finite beam width, the list of hypotheses is pruned only once at the very end—after all possible parsing hypotheses have been generated—instead of performing continuous pruning during decoding, resulting in higher accuracy guarantees.

We develop two variants of the decoder. The first one is for finding the most likely parsed string. The more realistic use case is where the input sequence simply serves as *advice* on where to find the top scoring parse under a secondary metric—i.e. where the element with the highest decoder score is *not necessarily* the one with the highest probability of occurring when sampled. In this variant the speedup becomes more pronounced (i.e. the runtime scales less and less quickly in the input length) the better the advice (i.e. the steeper the power law falloff of the input, see Fig. 1).

Our novel algorithmic contribution is to analyse a recently developed quantum maximum finding algorithm (Apeldoorn et al. 2017) and its expected runtime when provided with a biased quantum sampler that we developed for formal grammars, under the premise that at each step the input tokens follow a power-law distribution; for a probabilistic sequence obtained from Mozilla's *DeepSpeech* (which we show satisfies the premise), the quantum search decoder is a power of ≈ 4–5 faster than possible classically (Fig. 2).

In the following we assume basic familiarity with the notion of quantum computation, but provide a short overview for the reader in Appendix 1.

2 Main results

In this paper, we address the question of decoding a probabilistic sequence of words, letters, or generally tokens, obtained, e.g., from the final softmax layer of a recurrent neural network, or given as a probabilistic list of heuristic parse transitions. These models are essentially identical from a computational perspective. Hence, we give the following formal setup, and will speak of a decoding task, leaving implicit the two closely related applications.

Given an alphabet Σ, we expect as input a sequence of random variables $X = (X_1, X_2, \ldots, X_n)$, each distributed as $X_i \sim \mathcal{D}_i^{\Sigma}$. The distributions $\mathcal{D}_i^{\Sigma}$ can in principle vary for each i; furthermore, the X_i can either be independent, or include correlations. The input model is such that we are given this list of distributions explicitly, e.g. as a table of floating point numbers; for simplicity of notation we will continue to write X_i for such a table. The decoding machine M is assumed to ingest the input (a sample of the X_i) one symbol at a time, and branch according to some factor R at every step; for simplicity we will assume that R is constant (e.g. an upper bound to the branching ratio at every step). As noted, M can for instance be a parser for a formal grammar (such as an Earley parser (Earley 1970)) or

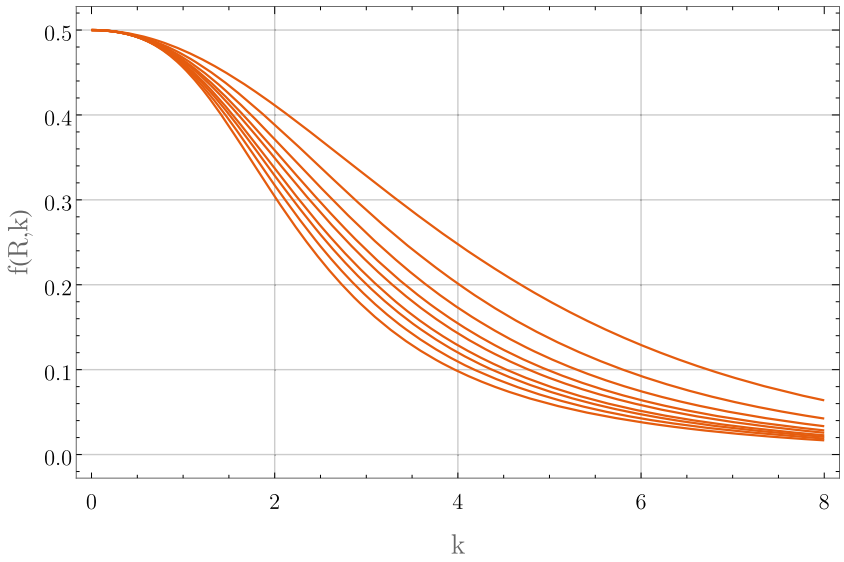


Fig. 1 Exponent $f(R, k)$ of expected runtime of QUANTUMSEARCHDECODE, when fed with a power law input with exponent k, over R alphabet tokens; plotted are individual curves for the values $R \in \{3, 5, 10, 15, 20, 30, 40, 60, 100\}$, from top to bottom. For all R, $f(R, k)$ drops off exponentially with growing k

some other type of language model; it can either accept good input strings, or reject others that cannot be parsed. The set of configurations of M that lead up to an accepted state is denoted by Ω; we assume that everything that is rejected is mapped by the decoder to some type of sink state $\omega \neq \Omega$. For background details on formal grammars and automata we refer the reader to Hopcroft et al. (2001), and we provide a brief summary over essential topics in Appendix 2.

While we can allow M to make use of a heuristic that attempts to guess good candidates for the next decoding step, it is not difficult to see that a randomised input setting is already more generic than allowing extra randomness to occur within M itself: we thus restrict our discussion to a decoder M that processes a token sequence step by step, and such that its state itself now simply becomes a sequence $(M_i)_{i \leq n}$ of random variables. More precisely, described as a stochastic process, the M_i are random variables over the set Ω of internal configurations after the automaton has ingested X_i, given that it has ingested $X_{i-1}, \ldots, X_1$ prior to that, with a distribution $\mathcal{D}_i^{\Omega}$. The probability of decoding

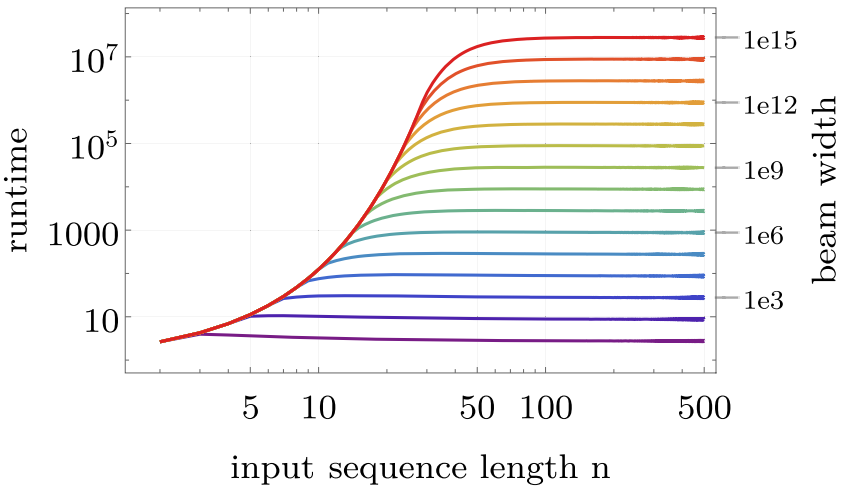


Fig. 2 Runtime of quantum beam search decoding the output of Mozilla's *DeepSpeech* LSTM with a grammar, assuming an average branching ratio of $R = 5$, a token power law distribution with exponent $k = 2.91$, and post-amplification of the quantum search decoder with a constant number of retained hypotheses $N_{\text{hyp}} \in \{10^1, \ldots, 10^{15}\}$, plotted in rainbow colors from purple to red, bottom to top. In the left region, where full QUANTUMSEARCHDECODING is performed (as the beam comprises all possible hypotheses), a super-Grover speedup is obtained (Corollary 2). Where the beam width saturates, a Grover speedup is retained, and hypotheses are pruned only *after* all hypotheses have been constructed

a specific accepted string $x = (x_1, \ldots, x_n)$ is then given by the product of the conditional probabilities

$$\begin{aligned}\Pr(M_n = x) &:= \mathcal{N}\Pr(X = x) \\ &= \frac{1}{\mathcal{N}} \prod_{i=1}^{n} \Pr(X_i = x_i | X_j = x_j,\, j \leq i-1) \end{aligned} \quad (1)$$

where $\mathcal{N} = \sum_{x \in \Omega} \Pr(X = x)$. In slight abuse of notation we write $M_n = x$ when we mean $M_n = y(x)$, where $y(x)$ is the configuration of the parser M that was provided with some input to produce the parsed string x (which is unambiguous as there is a one-to-one mapping between accepted strings and parser configurations $y(x)$). Similarly, we write $x \in \Omega$ for an accepted string/decoded path.

The obvious question is: which final accepted string of the decoder is the most likely? This is captured in the following computational problem.

MOST LIKELY PARSE

Input: Decoder M over alphabet Σ, set of accepting configurations Ω. Sequence of random variables $(X_i)_{i \leq n}$ over sample space Σ.

Question: Find $\sigma = \text{argmax}_{x \in \Omega} \Pr(M_n = x)$.

Classically, it is clear that if we have a procedure that can sample the random variable M_n efficiently, then we can find the most likely element with an expected runtime of $1/\Pr(M_n = \sigma)$, as this is the number of samples we are expected to draw to see the element once. While such sampling algorithms might be inefficient to construct in general, we emphasise that the question of drawing samples from strings over a formal language is an active field of research, and algorithms to sample *uniformly* are readily available for a large class of grammars: in linear time for regular languages (Bernardi and Giménez 2012; Oudinet et al. 2013), but also context-free grammars/restrictions thereof (McKenzie 1997; Goldwurm et al. 2001; Hickey and Cohen 1983; Gore et al. 1997; Denise 1996), potentially with global word bias (Reinharz et al. 2013; Lorenz and Ponty 2013; Denise et al. 2000; Ponty 2012).

In Theorem 3 and Section 3.1, we lift such a classical uniform sampler to a quantum sampler (denoted $\mathbf{U}_\mu$) with *local* (instead of global) word bias, which we can use to obtain a quantum advantage when answering MOST LIKELY PARSE. We note that the techniques used to prove Theorem 3 may well be used to obtain a (potentially faster) classical Monte Carlo procedure to sample from M_n. In what follows, we will therefore keep the decoder's time complexity separate from the sampler's runtime and simply speak of the decoder's query complexity to $\mathbf{U}_\mu$, but we emphasise that constructing such an $\mathbf{U}_\mu$ is efficiently possible, given a classical description of an automaton that parses the grammar at hand.

We start with the following observation, proven in Section 4.1.

Theorem 1 *For an input sequence of n random variables to a parser with sampling subroutine* $\mathbf{U}_\mu$*, there exists a quantum search algorithm answering* MOST LIKELY PARSE*, using* $\pi/4\sqrt{Pr(M_n = \sigma)}$ *queries to* $\mathbf{U}_\mu$.

As explained, this theorem formalises the expected quadratic speedup of the runtime as compared to a classical algorithm based on sampling from M_n. Given the input to the parser is power-law distributed (see Definition 1), this allows us to formulate the following corollary.

Corollary 1 *If the* $X_i \sim \text{Power}_R(k)$*, answering* MOST LIKELY PARSE *requires at most* $1/H_R(k)^{n/2}$ *queries; where* $H_R(k) = \sum_{i=1}^{R} i^{-k}$.

Yet a priori, it is not clear that the weight of a decoded path (e.g. the product of probabilities of the input tokens) also corresponds to the highest score we wish to assign to such a path. This becomes obvious in the setting of a heuristic applied to a live translation: while at every point in time the heuristic might be able to guess a good forward transition, it might well be that long range correlations strongly affect the likelihood of prior choices. Research addressing these long-distance "collocations" indicates that LSTM models are capable of using about 200 tokens of context on average, but that they sharply distinguish nearby context ($\approx$ 50 tokens) from the distant past. Furthermore, such models appear to be very sensitive to word order within the most recent context, but ignore word order in the long-range context (more than 50 tokens away) (Zhu et al. 2015; Dabrowska 2008; Khandelwal et al. 2018). Similarly, transformer-type architectures with self-attention—while outperforming LSTMs—feature a fixed-width context window; extensions thereof are an active field of research (Al-Rfou et al. 2019; Dai et al. 2019; Kitaev et al. 2020).

To address this setting formally, we assume there exists a scoring function $F : \Omega \longrightarrow \mathbb{R}$, which assigns scores to all possible decoded paths. Without loss of generality, there will be one optimal string which we denote with $\tau = \text{argmax}_{x \in \Omega} F(x)$. Furthermore, we order all decoded strings Ω in some fashion, and index them with numbers $i = 1, \ldots, |\Omega|$. Within this ordering, τ can now be in different places—either because the heuristic guesses differently at each step, or because the input sequence varied a little. We denote the probability that the marked element τ is at position i with p_i. In essence, the position where τ is found is now a random variable itself, with probability mass $\Pr(\text{finding } \tau \text{ at index } i) = p_i$.

For the decoder probabilities $\Pr(M_n = x)$ to serve as *good advice* on where to find the highest-score element under the metric F, we demand that the final distribution over the states of the decoder puts high mass where the

highest-scoring element often occurs; or formally that

$$\Pr(M_n = \text{string with index } i) = p_i. \tag{2}$$

To be precise, we define the following problem.

HIGHEST SCORE PARSE

Input: Decoder M over alphabet Σ and with state space Ω. Sequence of random variables $(X_i)_{i\leq n}$ over sample space Σ. Scoring function $F : \Omega \longrightarrow \mathbb{R}$.

Promise: Eq. (2).

Question: Find $\tau = \text{argmax}_{x\in\Omega} F(x)$.

What is the classical baseline for this problem? As mentioned in Montanaro (2011), if p_x is the probability that x is the highest-scoring string, then in expectation one has to obtain $1/p_x$ samples to see x at least once. Any procedure based on sampling from the underlying distribution p_x thus has expected runtime $\sum_{x\in\Omega} \frac{1}{p_x} \times p_x = |\Omega|$. In a sense this is as bad as possible; the advice gives zero gain over iterating the list item by item and finding the maximum in an unstructured fashion. Yet provided with the same type of advice, a quantum computer can exhibit tremendous gains over unstructured search, such as the following statement, formally proven in Section 4.2.

Theorem 2 *With the same setup as in* Theorem 1 *but under the promise that the input tokens are iid with* $X_i \sim \text{Power}_{|\Sigma|}(k)$ *over alphabet* Σ *(*Definition 1*), that the decoder has a branching ratio* $R \leq |\Sigma|$*, and that we can uniformly sample from the grammar to be decoded, there exists a quantum algorithm* QUANTUMSEARCHDECODE *(*Algorithm 1*) answering* HIGHEST SCORE PARSE *with an expected number of iterations*

$$\text{RT}_1(R,k,n) = \text{O}\left(R^{nf(R,k)}\right),$$
$$\text{where} \quad f(R,k) = \log\left(\frac{H_R(k/2)}{H_R(k)^{1/2}}\right)\Big/ \log R,$$

and where $H_R(k)$ *is defined in* Corollary 1.

There exists no classical algorithm to solve this problem based on taking stochastic samples from the decoder M that requires less than $\Omega(R^n)$ *samples.*

The exponent $f(R,k)$ indicates the speedup over a classical implementation of the decoding algorithm (which would have to search over R^n elements). We find that $f(R,k) < 1/2$ for all $R, k > 0$, and in fact $f(R,k) \longrightarrow 0$ exponentially quickly with k; we formulate the following corollary.

Corollary 2 *For* $k > 0$*,* QUANTUMSEARCHDECODE *is always faster than plain Grover search (with runtime* $\propto R^{n/2}$*); the extent of the speedup depends on the branching ratio R and the power law exponent k (see* Fig. 1*).*

Finally, in Section 5 we modify the full quantum search decoder by only searching over the paths with likelihood above some given threshold (that we allow to depend on n in some fashion), turning the decoder into a type of beam search, but where the pruning only happens at the very end (Algorithm 2). This means that in contrast to beam search, the top scoring element is found over the *globally* most likely parsed paths, avoiding the risk early beam pruning brings. We analyse the runtime of Algorithm 2 for various choices of beam width numerically, and analyse its performance on a concrete example—Mozilla's *DeepSpeech* implementation, a speech-to-text LSTM which we show to follow a power-law token distribution at each output frame (see Appendix 8 for an extended discussion). For *DeepSpeech*, we empirically find that input sequence lengths of up to 500 tokens can realistically be decoded, with an effective beam width of 10^{15} hypotheses—while requiring $\approx 3 \times 10^6$ search iterations (cf. Fig. 2). As expected, the super-Grover speedup from Corollary 2 is achieved in the regime where full QUANTUMSEARCHDECODING happens; once the beam width saturates, the speedup asymptotically approaches a quadratic advantage as compared to classical beam search.

3 Quantum search decoding

In this section, we give an explicit algorithm for QUANTUMSEARCHDECODE. As mentioned before (see Section 2), we assume we have access to a classical sampling algorithm that, given a list of transition probabilities determined by the inputs $X_1, \ldots, X_n$, yields a random sample drawn uniformly from the distribution. Since this sampler is given as a classical probabilistic program, we first need to translate it to a quantum algorithm. We start with the following lemma.

Lemma 1 *For a probabilistic classical circuit with runtime* $T(n)$ *and space requirement* $S(n)$ *on an input of length n, there exists a quantum algorithm that runs in time* $\text{O}(T(n)^{\log_2 3})$ *and requires* $\text{O}(S(n)\log T(n))$ *qubits.*

Proof Follows from Thm. 1 in Buhrman et al. (2001); see Appendix 7. □

3.1 Biased quantum sampling from a regular or context-free grammar

Given a sampler that can yield *uniformly* distributed strings s_i of a language, we want to raise it to a quantum circuit $\mathbf{U}_\mu$ that produces a quantum state which is a *biased* superposition over all such strings $s_i = a_{i1}a_{i2}\cdots a_{in}$, where each string is weighted by the probability p_{ij} of the symbol

a_{ij} occurring at index j (i.e. by Eq. (1)). In addition to the weighted superposition, we would like to have the weight of each state in the superposition spelled out as an explicit number in an extra register (e.g. as a fixed precision floating point number), i.e. as

$$\mathbf{U}_\mu|0\rangle = |\mu\rangle \propto \sum_{q\in\Omega} \sqrt{p_q}|h_q\rangle|p_q\rangle|q\rangle, \tag{3}$$

where Ω is the set of accepted strings reachable by the decoder in n steps, $|h_q\rangle$ is an ancillary state that depends on q and is contained in the decoder's work space, where q is a state reached by reading the input sequence $a_{q1}, a_{q2}, \ldots, a_{qn}$. The weights $p_q = \prod_{j=1}^{n} p_{qj}$.

As outlined in the introduction, we know there exist uniform classical probabilistic samplers for large classes of grammars, e.g. for regular languages in linear time (e.g. Oudinet et al. 2013) and polynomial time for variants of context free grammars (e.g. Goldwurm et al. 2001). Keeping the uniform sampler's runtime separate from the rest of the algorithm, we can raise the sampler to a biased quantum state preparator for $|\mu\rangle$.

Theorem 3 *Assume we are given a classical probabilistic algorithm that, in time $T(n)$, produce a uniform sample of length n from a language, and we are also given a list of independent random variables $X_1, \ldots, X_n$ with pdfs $p_{i,j}$ for $i = 1, \ldots, n$ and $j = [\Sigma]$. Then we can construct a quantum circuit $\mathbf{U}_{\mu'}$ that produces a state $|\mu'\rangle$ ϵ-close (in total variation distance) to the one in* Eq. (3)*. The algorithm runs in time $\mathrm{O}(T(n)^{1.6} \times n^3\kappa/\epsilon^2)$, where κ is an upper bound on the relative variance of the conditional probabilities $Pr(a|s_1 \ldots s_i)$, for $a, s_i \in \Sigma$, for the variable X_{i+1} given the random string $X_i X_{i-1} \cdots X_1$.*

Proof See Appendix 3. □

Getting a precise handle on κ strongly depends on the grammar to be parsed and the input presented to it; it seems unreasonable to claim any general bounds as it will most likely be of no good use for any specific instance. However, we note that it is conceivable that if the input is long and reasonably independent of the language to be sampled, then κ should be independent of n, and $\kappa \approx 1/p(r_{\min})$, where $p(r)$ is the distribution of the input tokens at any point in time—e.g. $p(r) \propto r^{-k}$ as in a power law.[1]

[1]This should make intuitive sense: the branching ratios are already biased with respect to the number of future strings possible with prefix s; if the input sequence is independent of the grammar, then we would expect them to weigh the strings roughly uniformly; the extra factor of $1/p(r_{\min})$ simply stems from the weighing of the token we bin by, namely a.

3.2 The quantum search decoder

The quantum algorithm underlying the decoder is based on the standard maximum finding procedure developed by Dürr and Høyer (1996) and Ahuja and Kapoor (1999), and its extension in Apeldoorn et al. (2017) used in the context of SDP solvers.

The procedure takes as input a unitary operator $\mathbf{U}_\mu$ which prepares the advice state, and a scoring function F which scores its elements, and returns as output the element within the advice state that has the maximum score under F. As in Section 3.1, we assume that F can be made into a reversible quantum circuit to be used in the comparison operation. We also note that reversible circuits for bit string comparison and arithmetic are readily available (Oliveira and Ramos 2007), and can, e.g., be implemented using quantum adder circuits (Gidney 2018).

Algorithm 1 Algorithm for quantum search decoding.

function QUANTUMSEARCHDECODE$_m$($\mathbf{U}_\mu$, F)
  $bestScore \leftarrow -\infty$, $counter \leftarrow 0$
  repeat
    // comparator against current best score
    $cmp \leftarrow [(\cdot) \mapsto (bestScore < \cdot)]$
    // amplify elements $\geq$ pivot
    $|\psi\rangle \leftarrow$ EXPONENTIALSEARCH($\mathbf{U}_\mu$, $cmp \circ F$)
    // measure new best score
    $bestScore \leftarrow \mathbf{M}_{\text{score}}|\psi\rangle$
    $counter \leftarrow counter + 1$
  until $counter = m$
end function

Algorithm 1 lists the steps in the decoding procedure. As a subroutine within the search loop, we perform exponential search with oblivious amplitude amplification (Berry et al. 2014). As in the maximum finding algorithm, the expected query count for quantum search decoding is given as follows.

Theorem 4 *If $x \in \Omega$ is the highest-scoring string and p_x its score in* Eq. (3)*, the expected number of iterations in* QUANTUMSEARCHDECODE *to find it maximum is* $\mathrm{O}(\min\{1/\sqrt{p_x}, \sqrt{n}\})$.

Proof Immediate by Apeldoorn et al. (2017). □

In the following we will for simplicity say $|x\rangle$ is the highest-scoring string (including ancilliary states given in Eq. (3)), and write $|\langle x|\mu\rangle|$ instead of $\sqrt{p_x}$. It is clear that the two notions are equivalent.

4 Power law decoder input

In this section we formally prove that if the decoder is fed independent tokens that are distributed like a power law, then the resulting distribution over the parse paths yields a super-Grover speedup—meaning the decoding speed is faster than applying Grover search, which itself is already quadratically faster than a classical search algorithm that traverses all possible paths individually.

A power law distribution is the discrete variant of a Pareto distribution, also known as Zipf's law, which ubiquitously appears in the context of language features (Jäger 2012; Stella and Brede 2016; Egghe 2000; Piantadosi 2014). This fact has already been exploited by some authors in the context of generative models (Goldwater et al. 2011).

Formally, we define it as follows.

Definition 1 Let A be a finite set with $|A| = R$, and $k > 1$. Then $\text{Power}_R(k)$ is the power law distribution over R elements: for $X \sim \text{Power}_R(k)$ the probability density function $\Pr(X = x) = r^{-k}/H_R(k)$ for an element of rank r (i.e. the r^{th} most likely element), where $H_R(k)$ is the R^{th} harmonic number of order k (Corollary 1).

We are interested in the Cartesian product of power law random variables, i.e. sequences of random variables of the form $(X_1, \ldots, X_n)$. Assuming the random variables $X_i \sim \text{Power}_R(k)$ are all independent and of rank r_i with pdf $q(r_i) = r_i^{-k}/H_R(k)$, respectively, it is clear that

$$p(r_1, \ldots, r_n) = \prod_{i=1}^{n} q(r_i) = \frac{1}{H_R(k)^n} \frac{1}{(r_1 \cdots r_n)^k}. \tag{4}$$

As in Montanaro (2011), we can upper bound the number of decoder queries in QuantumSearchDecode by calculating the expectation value of the iterations necessary—given by Theorem 4—with respect to the position of the top element.

We assume that at every step, when presented with choices from an alphabet Σ, the parsed grammar branches on average $R \leq |\Sigma|$ times. Of course, even within a single time frame, the subset of accepted tokens may differ depending on what the previously accepted tokens are. This means that if the decoder is currently on two paths β_1 (e.g. corresponding to "I want") and β_2 ("I were"), where the next accepted token sets are $\Sigma_1, \Sigma_2 \subset \Sigma$ (each different subsets of possible next letters for the two presented sentences), respectively, then we do *not* necessarily have that the total probability of choices for the two paths—$\Pr(\Sigma_1)$ and $\Pr(\Sigma_2)$—are equal. But what does this distribution over all possible paths of the language, weighted by Eq. (1), look like?

Certainly this will depend on the language and type of input presented. Under a reasonable assumption of independence between input and decoded grammar, this becomes equivalent to answering the following question: let X be a product-of-powerlaw distribution with pdf given in Eq. (4), where every term is a powerlaw over Σ. Let Y be defined as X, but with a uniformly random subset of elements deleted; in particular, such that R^n elements are left, for some $R < |\Sigma|$. Is Y distributed as a product-of-powerlaws as in Eq. (4), but over R elements at each step? In the case of continuous variables this is a straightforward calculation (see Appendix 5); numerics suggest it also holds true for the discrete case.

But even if the input that the parser given is independent of the parsed grammar, it is not clear whether the *sample distribution* over R (i.e. sampling R out of $|\Sigma|$ power-law distributed elements) follows the same power law as the original one over Σ; this is in fact not the case in general (Zhu et al. 2015). However, it is straightforward to numerically estimate the changed power law exponent of a sample distribution given R and $|\Sigma|$—and we note that the exponent shrinks only marginally when $R < |\Sigma|$.

In this light and to simplify the runtime analysis, we therefore assume the decoder accepts exactly R tokens at all times during the parsing process (like an R-ary tree over hypotheses) with a resulting product-of-powerlaw distribution, and give the runtimes in terms of the branching ratio, and not in terms of the alphabet's size. This indeed yields a fair runtime for comparison with a classical variant, since any classical algorithm will *also* have the aforementioned advantage (i.e. we assume the size of final elements to search over is R^n, which precisely corresponds to the number of paths down the R-ary tree).

4.1 Most likely parse: query bound

In this case F simply returns p_q as the score in Eq. (3). If x labels the highest-mass index of the probability density function (neglecting the ancilliary states in Eq. (3) for simplicity), it suffices to calculate the state overlap $|\langle x|\mu\rangle|$. By Eq. (4), we then have $|\langle x|\mu\rangle|^2 = H_R^{-n}(k)$. The claim of Corollary 1 follows from these observations.

4.2 Highest score parse: simple query bound

We aim to find a top element scored under some function F under the promise that $|\mu\rangle$ (given in Eq. (3)) presents good advice on where to find it, in the sense of Eq. (2). The expected runtimes for various power law falloffs k can be obtained by taking the expectation with respect to p_x as in Montanaro (2011).

In order to do so, we need to be able to calculate expecation values of the cartesian product of power law random variables, where we restrict the domain to those

elements with probability above some threshold. We start with the following observation.

Lemma 2 *If* QUANTUMSEARCHDECODE *receives as input iid random variables* $X_1, \ldots, X_n$, *with* $X_i \sim \text{Power}_R(k)$, *then the number of queries required to the parser is* $\text{RT}_1(R,k,n) = \text{O}\left(H_R(k/2)^n / H_R(k)^{n/2}\right)$.

Proof The expectation value of $1/\langle x|\mu\rangle$ is straightforward to calculate; writing $\boldsymbol{r} = (r_1, \ldots, r_n)$, by Eq. (4), we have

$$\begin{aligned}\mathbb{E}(1/\langle x|\mu\rangle) &= \sum_{\boldsymbol{r}} p(\boldsymbol{r}) \times \frac{1}{\sqrt{p(\boldsymbol{r})}} \\ &= \frac{1}{H_R(k)^{n/2}} \sum_{r_1=1}^{R} \cdots \sum_{r_n=1}^{R} \frac{1}{(r_1 \cdots r_n)^{k/2}}.\end{aligned}$$

As $\text{O}(\min\{1/\langle x|\mu\rangle, \sqrt{n}\}) \leq \text{O}(1/\langle x|\mu\rangle)$ the claim follows. □

We observe that the runtime in Lemma 2 is exponential in n. Nevertheless, as compared to a Grover algorithm—with runtime $R^{n/2}$—the base is now dependent on the power law's falloff k. We can compare the runtimes if we rephrase $\text{RT}_1(R,k,n) = R^{nf(R,k)}$, by calculating

$$\begin{aligned}&\left(\frac{H_R(k/2)}{H_R(k)^{1/2}}\right)^n = R^{nf(R,k)} \\ \iff & f(R,k) = \log\left(\frac{H_R(k/2)}{H_R(k)^{1/2}}\right) \Big/ \log R.\end{aligned}$$

We observe that the exponent $f(R,k) \in (0, 1/2)$, i.e. it is always faster than Grover, and always more than quadratically faster than classically. The exponent's precise dependency on k for a set of alphabet sizes R is plotted in Fig. 1. For growing k, $f(R,k)$ falls off exponentially.

4.3 MOST LIKELY PARSE: full query bound

A priori, it is unclear how much we lose in Lemma 2 by upper-bounding $\text{O}(\min\{1/\langle x|\mu\rangle, \sqrt{n}\})$ by $\text{O}(1/\langle x|\mu\rangle)$—so let us be more precise. In order to evaluate the expectation value of the minimum, we will break up the support of the full probability density function $p(\boldsymbol{r})$ into a region where $p(\boldsymbol{r}) > 1/R^n$, and its complement. Then, for two constants C_1 and C_2, we have for the full query complexity

$$\begin{aligned}\text{RT}_2(R,k,n) &= \mathbb{E}\left[\text{O}(\min\{1/\langle x|\mu\rangle, \sqrt{n}\})\right] \\ &= C_1 \sum_{\boldsymbol{r}: p(\boldsymbol{r}) > 1/R^n} \sqrt{p(\boldsymbol{r})} + C_2 \sqrt{n} \sum_{\boldsymbol{r}: p(\boldsymbol{r}) \leq 1/R^n} p(\boldsymbol{r}).\end{aligned} \tag{5}$$

In order to calculate sums over sections of the pdf $p(\boldsymbol{r})$, we first move to a truncated Pareto distribution by making the substitutions

$$\sum_{r \in A} \frac{1}{r^k} \longrightarrow \int_A \frac{1}{r^k} \mathrm{d}r, \quad H_R(k) \longrightarrow h_R(k) := \int_1^R \frac{1}{r^k} \mathrm{d}r.$$

While this does introduce a deviation, its magnitude is minor, as can be verified numerically throughout (see Fig. 4, where we plot both RT_1 and the continuous variant $\text{RT}_{1'}(R,k,n) := h_R^n(k/2) / h_R^{n/2}(k)$).

The type of integral we are interested in thus takes the form

$$M_{c,n}^{R,k_1,k_2} := \frac{1}{h_R^n(k_1)} \iiint_1^R \frac{\chi(r_1 \cdots r_n \leq c)}{(r_1 \cdots r_n)^{k_2}} \mathrm{d}r_1 \cdots \mathrm{d}r_n, \tag{6}$$

where k_1 is not necessarily equal to k_2, and typically $c = (R/h_R(k_1))^{n/k_1}$, which would reduce to the case we are seeking to address in Eq. (5). Here, $\chi(\cdot)$ denotes the characteristic function of a set, i.e. it takes the value 1 where the premise is true, and 0 otherwise. We derive the following closed-form expression.

Lemma 3 *For* $k \neq 1$, Eq. (6) *becomes*

$$\begin{aligned}M_{c,n}^{R,k_1,k_2} = \frac{(-1)^n}{k'^n h_R^n(k_1)} &\sum_{j=0}^{\min\{n, \lfloor c'/a' \rfloor\}} \binom{n}{j} \\ &\times \left(\mathrm{e}^{a'k'j} - \mathrm{e}^{-c'k'} \sum_{l=0}^{n-1} \frac{(a'k'j - c'k')^l}{l!}\right),\end{aligned}$$

where $k' = 1 - k_2$, $c' = \log c$, $a' = \log R$.

Proof See Appendix 4. □

5 Quantum beam search decoding

The goal of this section is to modify the QUANTUMSEARCHDECODER such that it behaves more akin to a classical beam search algorithm. More specifically, instead of searching for the top scored element which could sit *anywhere* within the advice distribution, we make the assumption that wherever the advice probability lies below some threshold $p(x) < p_0$—where p_0 can be very small—we discard those hypotheses. This is done by dovetailing a few rounds of amplitude amplification to suppress all beam paths with probability less than p_0 (which we can do, since we have those probabilities written out as numbers within the advice state $|\mu\rangle$ in Eq. (3)); a schematic of the algorithm can be found in Algorithm 2.

Of course we only want to do this if the number of amplification rounds, given as the squareroot of the inverse of the leftover probability $\sum_{x:p(x)\geq p_0} p(x)$, is small (i.e. constant, or logarithmic in n). We note that this expression is, as before, well-approximated by $M^{R,k,k}_{p_0,n}$ given in Lemma 3.

Algorithm 2 Algorithm for beam search decoding.

function QUANTUMBEAMDECODE$_m(\mathbf{U}_\mu, F, p_0)$
  $bestScore \leftarrow -\infty$, $counter \leftarrow 0$
  repeat
    // comparator against threshold
    $cmp_1 \leftarrow [(\cdot) \mapsto (p_0 < \cdot)]$
    // comparator against current best score
    $cmp_2 \leftarrow [(\cdot) \mapsto (bestScore < \cdot)]$
    // prune hypotheses
    $amp \leftarrow [(\cdot) \mapsto$ AMPLITUDEAMPLIFY$(\cdot, cmp_1)]$
    // select elements $\geq$ pivot
    $|\psi\rangle \leftarrow$ EXPONENTIALSEARCH$(amp \circ \mathbf{U}_\mu, cmp_2 \circ F)$
    // measure new best score
    $bestScore \leftarrow \mathbf{M}_{\text{score}}|\psi\rangle$
    $counter \leftarrow counter + 1$
  until $counter = m$
end function

In beam search, only the top scoring hypotheses are kept around at any point in time; the difference to our method is of course that we can score the elements *after every hypothesis has been built*. This is not possible in the classical case, since it would require an exponential amount of memory, or postselection. As in Section 3, we have the two cases of finding the top scoring path and the most likely parse. Deriving a runtime bound for MOST LIKELY PARSE is straightforward—and does not, in fact, gain anything. This is because when finding the maximum-likelihood path τ, one performs amplitude amplification on that element anyhow, and $p(\tau) > p_0$—so it is within the set of elements with probability kept intact by the post-amplification.[2]

The only interesting case of amplifying the advice state in QUANTUMSEARCHDECODE to raise it to a beam search variant is thus for the case of HIGHEST SCORE PARSE, using the decoder's output as advice distribution. Instead of listing a series of results for a range of parameters, we provide an explicit example of this analysis with real-world parameters derived from Mozilla's DeepSpeech neural network in the next section, and refer the reader to Appendix 6 for a more in-depth analysis of variants of a constant and non-constant amount of post-amplification.

[2] If anything, p_0 introduces some prior knowledge about the first pivot to pick for maximum finding.

6 DeepSpeech

6.1 Analysis of the output rank frequency

To support the applicability of our model, we analysed our hypothesis that the output probabilities of an LSTM used to transcribe voice to letters—which can then be used, e.g., in a dialogue system with an underlying parser—is distributed in a power-law like fashion. More specifically, we use *DeepSpeech*, Mozilla's implementation of Baidu's *DeepSpeech* speech recognition system (Hannun et al. 2014; Mozilla 2019b); our hypothesis was that these letter probabilities follow a power-law distribution; our data supports this claim (see Appendix 8, also for a discussion of the LSTM's power-law output—a model feature—vs. the power-law nature of natural language features).

6.2 Runtime bounds for quantum beam search decoding

We take the power law exponent derived from Mozilla's DeepSpeech neural network, $k = 3.03$ (cf. Appendix 8), and derive runtime bounds for decoding its output with a parser under the assumption that, on average, we take $R = 5$ branches in the parsing tree at every time step. As discussed in Section 4, the sampling distribution over five elements only yields a slightly lower exponent of $k = 2.91$. How does quantum beam search perform in this setting, and how many hypotheses are actually searched over? And what if we fix the beam's width to a constant, and increase the sequence length? We summarise our findings in Figs. 2, 7 and 8).

7 Summary and conclusions

We have presented a quantum algorithm that is modelled on and extends the capabilities of beam search decoding for sequences of random variables. Studies of context sensitivity of language models have shown that state-of-the-art LSTM models are able to use about 200 tokens of context on average while working with standard datasets (WikiText2, Penn Treebank) (Khandelwal et al. 2018); state of the art transformer-based methods level off at a context window of size 512 (Al-Rfou et al. 2019). On the other hand, under the premise of biased input tokens, our quantum search decoding method is guaranteed to find—with high constant success probability—the global optimum, and it can do so in expected runtime that is always more than quadratically faster than possible classically. As demonstrated empirically (cf. Fig. 2), our quantum beam search variant features a runtime independent of the sequence length: even for token sequences of length > 500

the top 10^{14} global hypotheses can be searched for an optimal prediction, within 10^7 steps.

We have further shown that neural networks used in the real world—concretely *DeepSpeech*—indeed exhibit a strong power law distribution on their outputs, which in turn supports the premise of our algorithm; how the performance scales in conjunction with a native recurrent quantum neural network such as in Bausch (2020) is an interesting open question.

Appendix 1. Quantum computing: preliminaries and notation

In this section we briefly review the basic notions and notations in quantum computation, referring to Nielsen and Chuang (2010) for more details.

The usual unit of classical computation is the bit, a Boolean variable taking values in $\mathbb{Z}_2 = \{0, 1\}$. Its analogue in quantum computation is called the qubit, and represents the state of a physical quantum 2-level system. A qubit can take values in $\mathbb{C}^2$, i.e. linear combinations or superpositions of two classical values (complex numbers)

$$\alpha|0\rangle + \beta|1\rangle$$

In particular we require that $|\alpha|^2 + |\beta|^2 = 1$. We have also introduced the Dirac bracket in the above:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle := \begin{pmatrix} 0 \\ \mathbb{1} \end{pmatrix}.$$

More generally, the set of states an m-qubit quantum register can take is the set of unit vectors

$$|\phi\rangle = \sum_{i \in \{0,1\}^m} \alpha_i |i\rangle \text{ with } \alpha_i \in \mathbb{C}, \text{ such that } \sum_i |\alpha_i|^2 = 1 \tag{7}$$

in the Hilbert space spanned by a set of orthonormal basis vectors $\{|i\rangle, i \in \{0, 1\}^m\}$, known as the *computational basis*. Each α_i is called the *amplitude of basis state* $|i\rangle$. We interpret the vector $|i\rangle$ as the m-dimensional complex vector v_i with entries given by $(v_i)_j = \delta_{ij}$, and also interchangeably as the integer i or the bit string that gives its binary representation $b_1 \dots b_m$ where b_i is either 0 or 1. Furthermore, and of key importance to quantum mechanics and computation, the vector $|b_1 \dots b_m\rangle \in \mathbb{C}^{2^m}$ is interpreted as a tensor product

$$|b_1 \dots b_m\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_m\rangle$$

of the m vectors $|b_i\rangle$ in $\mathbb{C}^2$. The $\otimes$ is often dropped for convenience, and we write $|b_1\rangle|b_2\rangle$ for $|b_1\rangle \otimes |b_2\rangle$.

Unitary operators There are two ways in which we can compute on a state ϕ. The first is by *unitary evolution* of the system under the Schrödinger equation with a specified Hamiltonian operator H

$$i\frac{d|\psi\rangle}{dt} = H|\psi\rangle,$$

where H is a hermitian matrix. Closed systems undergo reversible dynamics in quantum mechanics, and this dynamics is represented by unitary matrices. Since we can think of $|\phi\rangle$ as a vector in $\mathbb{C}^{2^m}$, a computation is represented by multiplication of this state by a $U \in \mathrm{SU}(2^m)$, i.e. $|\phi_{\text{out}}\rangle = U|\phi_{\text{in}}\rangle$. Recall that a matrix U is said to be unitary if $UU^\dagger = \mathbb{1}$, where $U^\dagger$ is the conjugate transpose of U. It is possible to compile a large 'algorithm' U down into elementary unitary operations, or quantum gates.

Measurements The second kind of operation we can perform on $|\phi\rangle$ is measurement. For our purposes, note that the postulates of quantum mechanics say that on measuring the state $|\phi\rangle$ in Eq. (7) *in the basis* $\{|i\rangle\}$, we obtain as outcome the basis state $|i\rangle$ with probability $|\alpha_i|^2$. Since we have chosen states to be normalised, the measurement gives a valid probability mass function over the set of classical m-bit strings. After the measurement, the state "collapses" to the observed basis state $|i\rangle$, and no further information can be retrieved from the original state.

Input models We will use two kinds of input models. The first is a quantum analogue of the classical query model, where inputs are accessed via a black-box or oracle that can be queried with an index i and returns the i-th bit of the input bit string. For a bit string $x \in \{0, 1\}^n$ we assume access to a unitary $\mathcal{O}_x$ which performs the map

$$\mathcal{O}_x|i\rangle|b\rangle|z\rangle = |i\rangle|b \oplus x_i\rangle|z\rangle, \tag{8}$$

where the first register consists of $\lceil \log n \rceil$ qubits, the second is a single qubit register to store the output of the query, and the third is any additional workspace the quantum computer might have and is not affected by the query. Here $\oplus$ is addition on $\mathbb{Z}_2$, i.e. the XOR operation in Boolean logic. Note that $\mathcal{O}_x$ can be used by a quantum computer to make queries in superposition:

$$\mathcal{O}_x\left(\frac{1}{n}\sum_{i=1}^{n}|i\rangle|b\rangle|z\rangle\right) = \frac{1}{n}\sum_i |i\rangle|b \oplus x_i\rangle|z\rangle, \tag{9}$$

Complexity measures For many theoretical studies in complexity theory, the query input model is a powerful setting where several results have been proven. In this model, the total number of queries made to the input oracle is the primary measure of algorithmic complexity, known as the *query complexity*.

For practical purposes, it is more important to understand the number of elementary quantum gates used to implement the unitary circuit corresponding to the algorithm in the

quantum circuit model. This is known as the *gate complexity* of the algorithm. The depth of the circuit is directly related to the time complexity, and gives an idea of how parallelisable the algorithm is.

Appendix 2. Languages, automata, and parsing

We refer the reader to the excellent overview of automata theory in Hopcroft et al. (2001). In the following, we introduce the two lowest-level type automata in the Chomsky hierarchy; those that accept regular languages, which are relatively restricted sets of strings, and those that accept context-free languages. While the latter are still somewhat restrictive, they are interesting in the context of natural language processing (NLP) as they are able to capture the essential features of a large fraction of human languages, and are as such a natural model to study in this context.

More pragmatically, we study these two types of languages because there exist efficient algorithms for executing automata that accepts them—in linear time in the length of the input string and of the regular expression in the case of a regular language, and in cubic time for the context-free case.

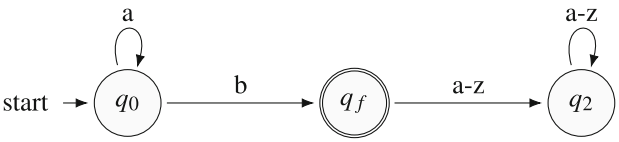
Moving further up in the Chomsky hierarchy, we first encounter context sensitive languages. Those have a natural associated automaton which is a linear space bounded nondeterministic Turing machine. These languages can be recognised by non-deterministic Turing machines with linear space, and so are in NSPACE(O(n)) for an input of size n. While the necessary memory might be bounded, we do not have good runtime guarantees (i.e. the Turing machine can potentially run for a time $\propto \exp(n)$). These models thus seem less relevant in practice; naturally, this just becomes worse when we allow an arbitrary Turing machine for which we have even less of a grasp on their runtime for a given string, and where even the question of whether the machine halts on some input is undecidable.[3]

Furthermore, for both NFAs and PDAs, at each step one symbol of the input is ingested. This symbol will then dictate how the automaton proceeds. If this input is not given by a fixed string but by a sequence of random variables over an alphabet, the sequence of states the automaton passes through as it processes the input will also be a sequence of random variables. Since we ingest one input symbol at every step, the random variable representing the state of the automaton at any given step will be significantly easier to analyse (in contrast to Turing machines, which can choose to ignore the given input; cf. Remark 1).

2.1 Regular languages and finite state automata

To give motivation for the study of finite state machines (FSAs), we start with a simple example of a regular expression, which is a string of characters from an alphabet (e.g. the letters "a"-"z"), with placeholders for arbitrary symbols (commonly denoted with a `.`), as well as quantifiers (e.g. the Kleene star `*`). For instance, the regular expression `hello` only matches the word "hello", whereas a regular expression `a*b` would match a string conprising any number of "a"s (including zero), and ending with "b", such as "aaaaab", or "ab", or "b".

Example 1 Take the regular expression `a*b` over the lowercase letters of the English alphabet. Then a finite state automaton that checks whether a string matches the regular expression would be



where the double lines of state 1 signify an accepting state. Given the string "aaab", one starts in the initial state q_0 and ingests all "a"s at the beginning of the string; once "b" is encountered, we jump to the accepting state q_f. Similarly, had the string been "aaabc", there would now be a "c" left to be ingested and the finite state automaton proceeds to state q_2. Since q_2 is not an accepting state, the regular expression does not match the string.

Notice that this model is deterministic: the automaton does not use randomness in making transitions, or indeed at any step at all. Any regular expression can be expressed as a finite state automaton—they are in fact equivalent, in the sense that both accept regular languages (Hopcroft et al. 2001, sec. 3.2).

With our newly gained intuition, we can now naturally define deterministic finite state automata as follows.

Definition 2 (Deterministic Finite State Automaton (DFA)) A deterministic finite state automaton is a tuple $(Q, \Sigma, \delta, q_0, F)$ of a finite set of states Q, an alphabet Σ, an initial state $q_0 \in Q$ and a set of accepting states $F \subseteq Q$. The allowed transitions are described by the transition function $\delta : Q \times \Sigma \longrightarrow Q$.

While regular expressions and FSAs are equivalent, there exist pathological examples for which enumerating

[3] Even for linear space bounded nondeterministic TMs, the halting problem is in principle decidable, as for a finite number of internal states any computation must eventually halt or loop; see Minsky (1967).

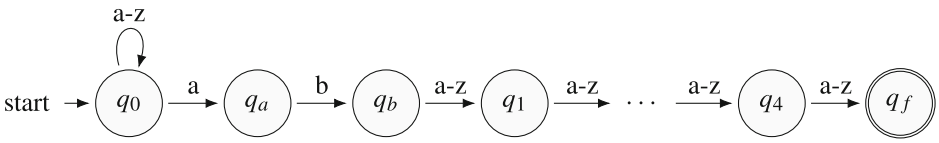
the possible intermediate states incurs an exponential overhead.

Example 2 Take the regular expression `.*ab.....`—which, written in this way, requires ten characters to note down. It is easy to see that this regular expression matches any string where the three characters preceding the five last characters equal "ab". However, since for every state $q \in Q$ the function δ allows precisely *one* forward transition per letter in the input string, the DFA has to memorise all the past characters it has seen, and thus requires a number of states that is exponential in the length of the regular expression (in an example such as the above) and/or exponential in the input length (e.g. for an expression like `.*ab.*`), see Hopcroft et al. (2001, sec. 2.3.6]).

To circumvent such a scenario, it would be beneficial to introduce the possibility of branching in order to attempt multiple matches in parallel, i.e. to have a list of multiple head states at which to attempt the next match. This is formalised with a nondeterministic finite state automaton, defined as follows.

Definition 3 (Nondeterministic Finite State Automaton (NFA)) An NFA is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 and F are defined as in Definition 2, while now the transition function allows a subset of Q to as possible target states for any transition, i.e. $\delta : Q \times \Sigma \longrightarrow 2^Q$.

Example 3 An NFA for the regular expression `.*ab.....` is given by



The initial state q_0 is set up such that it will always be active, while—as soon as an "a" is encountered—there is the possibility of branching out to q_a to explore whether the successive letters match the rest of the regular expression.

Note that in Example 3, as depicted, we dropped the non-accepting states after q_f; we simply implicitly assume for NFAs that every state $q \in Q$ has a link to a non-accepting state for all the missing transitions not yet present.

There is one final variant of nondeterministic automata we wish to define, which are called ϵ-NFA because they allow so-called ϵ-transitions, which need not be conditioned on reading an input symbol.

Definition 4 (ϵ-NFA) An ϵ-NFA is an NFA with the addition of ϵ-transitions, i.e. transitions that can be taken unconditionally.

NFAs or ϵ-NFAs appear intrinsically more powerful than DFAs, as they give more freedom in the way the parsing tree is traversed. However, this is not the case.

Theorem 5 ((Ullman et al. 1997; Kleene 1956; Louden 1997; Rabin and Scott 1959)) *The languages recognisable by NFAs, DFAs, and regular expressions, are all equal, and are called regular languages.*

It is known (Thompson 1968) that a regular language can be parsed in time O(mn), where m is the length of the regular expression, and n is the length of the input string.[4]

So what is the configuration of an NFA at any one point? In contrast to a DFA, the internal state the automaton in Example 3 is not anymore a single state $q \in Q$, but it will have multiple hypotheses: for instance, for an input "xyaaxabx...", the successive state of the automaton from Example 3 will be

```
x: [q0]
y: [q0]
a: [q0, qa]          # hypothesis q0 branched
a: [q0, qa]          # hypothesis qa was invalid, hypothesis q0 branched
x: [q0]              # hypothesis qa was invalid
a: [q0, qa]
b: [q0, qb]
x: [q0, q1]
...
```

[4] Curiously, even though the linear runtime of Thompson et al.'s NFA has been known since 1968, even until relatively recently, modern regular expression implementations were implemented using backtracking and memorisation techniques that have an exponential asymptotic runtime (Cox 2007).

At every time step, the set of hypotheses is the union of all the hypotheses that δ yields, when applied to the current list of marked vertices.

To be precise and because we will need it later we define the set of possible configurations for finite state automata as follows.

Definition 5 (DFA and NFA State Space) The set of possible states for a DFA as defined in Definition 2 is Q. The set of possible states for an NFA and ϵ-NFA is defined as $\Omega \subseteq 2^Q$, such that $c \in \Omega$ iff there exists an input string $s \in \Sigma^*$ such that the automaton, starting in its initial configuration is in state c after ingesting it.

Note that for DFAs we implicitly assumed that all such states Q are reachable from its initial configuration q_0, and a suitably chosen input.

We finally remark the following.

Remark 1 DFAs, NFAs, and ϵ-NFAs all ingest one input symbol per step. If there is an ϵ transition, the instantaneous state of an ϵ-NFA is that of *all* the states in Q reachable with ϵ transitions at any given moment.

In particular, what Remark 1 exemplifies is that there is no notion of "ϵ-loops"; if there is such a loop between two $q_1, q_2 \in Q$, the state of the automaton is in both q_1 and q_2 at the same time.

2.2 Context-free grammars and pushdown automata

While regular languages play an important role in computer science, it is straightforward to come up with an example of a language that cannot be parsed with an NFA, e.g. palindromes, or all strings of the form $a^n b^n$ for $n \in \mathbb{N}$. Regular grammars are the most restrictive ones in the Chomsky hierarchy (Chomsky 1956), i.e. they are *Type-3 grammars*.

The palindrome example can be captured by going one level further up in the hierarchy, to *Type-2* or *context-free grammars* (CFGs). They form a strict superset of regular languages and encompass common structures of modern programming languages such as nested balanced opening and closing brackets, e.g. { ( ) ([. . .] . . .) }.

While CFGs find wide application, e.g. for parsing (Hopcroft et al. 2001, sec. 5.3.2) or markup languages (Hopcroft et al. 2001, sec. 5.3.3), they still fail to capture more complicated language features, even ones as fundamental as requiring that variables have to be declared before they can be used within a scope; in fact, most widely used programming languages are not described with a context-free grammar. Indeed, the fact whether a programming language is Turing complete is independent on where its syntax is placed in the Chomsky hierarchy; the programming language "whitespace" can be described with a regular expression, whereas any language using brackets can easily be seen to require a stack, which goes beyond the capabilities of an NFA.

Instead of digressing too far into the theory of languages, we follow the same pattern as in Appendix 2.1 and discuss the natural machine that comes with CFGs. As mentioned, the existence of a stack seems to be necessary to recognise a context-free grammar, and in fact it also turns out to be sufficient.

Definition 6 (Pushdown Automaton (PDA) A pushdown automaton is an ϵ-NFA with access to a stack, which itself is given by a stack alphabet Γ with initial stack configuration $(Z_0) \in \Gamma$, and such that the transition operation

$$\delta : Q \times (\{\epsilon\} \cup \Sigma) \times \Gamma \longrightarrow 2^{Q \times \Gamma^*}$$

always pops the top element $a \in \Gamma$ off the stack and returns a list of pairs $(q, s) \in Q \times \Gamma^*$ that indicate what state to transition to, and what string to push onto the stack. Here Γ^* is the set of strings of arbitrary length in the stack alphabet, and ϵ is the empty input symbol.

Theorem 6 ((Hopcroft et al. 2001, sec. 6.3)) *Context-free grammars are equivalent to pushdown automata.*

As for the case of NFAs, there exist efficient algorithms based on dynamic programming such as an Earley parser that can decide whether a CFG accepts an input string, namely in time $O(n^3)$ for a string of length n (Earley 1970; Younger 1967).

As in Definition 5, we define the set of possible configurations that a pushdown automaton can be in as follows.

Definition 7 (PDA State Space) A PDA M's state space $\Omega \subset 2^{Q \times \Gamma^*}$ is the set of all those configurations of M reachable from its initial configuration $(q_0, (Z_0))$ with a suitably chosen input string $s \in \Sigma^*$.

We point out that an instantaneous state of a PDA can be a larger list than an instantaneous state of an NFA, since for any two possible chosen paths towards a state $q \in Q$, depending on the path the stack content can vary. We further note that the stack size is, in principle, unlimited, but as the transition function is specified a priori and is of finite size, the amount of information that can be pushed onto the stack for each step is upper-bounded by a constant as well; the stack can thus grow at most linearly in the runtime of the PDA.

Appendix 3. Biased quantum sampling from a regular or context-free grammar

In this section we rigorously proof Theorem 3, which we restate for completeness.

Theorem 6 *Assume we are given a classical probabilistic algorithm that, in time $T(n)$, produces uniform samples of length n from a language, and we are also given a list of independent random variables $X_1, \ldots, X_n$ with pdfs $p_{i,j}$ for $i = 1, \ldots, n$ and $j = [\Sigma]$. Then we can construct a quantum circuit $\mathbf{U}_{\mu'}$ that produces a state $|\mu'\rangle$ ϵ-close to the one in* Eq. (3)*. The algorithm runs in time $\mathrm{O}(T(n)^{1.6} \times n^3\kappa/\epsilon^2)$, where κ is an upper bound on the relative variance of the conditional probabilities $Pr(a|s_1 \ldots s_i)$, for $a, s_i \in \Sigma$, for the variable X_{i+1} given the random string $X_i X_{i-1} \cdots X_1$.*

Proof Using Lemma 1, translate the parser—which takes its input step by step—into a sequence of unitaries $\mathbf{U} = \mathbf{U}_n \cdots \mathbf{U}_1$. Considering a single unitary $\mathbf{U}_i$ at the i^{th} step, it is clear that it can be broken up into a family of unitaries $(\mathbf{U}_i^a)_{a\in\Sigma}$, such that each $\mathbf{U}_i^a$ is a specialisation of $\mathbf{U}_i$ when given a fixed input symbol $a \in \Sigma$. We define $\mathbf{V}_i^a$ to perform $\mathbf{U}_i^a$, and in addition store the input a in some ancillary workspace, e.g. via $\mathbf{V}_i^a|\phi\rangle|\xi\rangle = (\mathbf{U}_i^a|\phi\rangle)|\xi \oplus a\rangle$. Then define the block-diagonal unitary $\mathbf{V}_i := \mathrm{diag}(\mathbf{V}_i^a)_{a\in\Sigma}$, which acts like a controlled matrix, meaning that if $\mathbf{V}_i$ acts on some state $|\psi\rangle = |a\rangle|\phi\rangle$, then $\mathbf{V}_i|\psi\rangle = |a\rangle\mathbf{V}_i^a|\phi\rangle$. Naturally this works in superposition as well, e.g. $\mathbf{V}_i(\alpha|a\rangle + \beta|b\rangle)|\phi\rangle = \alpha|a\rangle\mathbf{V}_i^a|\phi\rangle + \beta|b\rangle\mathbf{V}_i^b|\phi\rangle$. We further assume that the $\mathbf{V}_0^a$ take as initial state $|0\rangle|q_0\rangle$.

The final step in augmenting the parser is to extend $\mathbf{V}_i$ to carry out a controlled multiplication: for a finite set of numbers $F \subset \mathbb{R}$ (e.g. fixed precision), and $d_1, d_2 \in F$, we write $\mathbf{V}_i(d_1)|a\rangle|d_2\rangle|\phi\rangle = |a\rangle|d_1 \times d_2\rangle\mathbf{V}_i^a|\phi\rangle$. We denote this extended unitary for step i with $\mathbf{U}_i'$.

The next ingredient we take is the classical uniform language sampler. Once again using Lemma 1, we raise it to a unitary $\mathbf{W}$, which takes as input a prefix $s_m := a_1 \cdots a_m$ of the m previously seen tokens, and a list of distributions over the future weights $W_m := (p_{i,j})_{m<j\le n}$. These are the distribution of tokens for each of the X_j. We then augment $\mathbf{W}$ to a circuit $\mathbf{W}'$ that quantumly performs the following classical calculations, in superposition over its input:

1. Draw S samples uniformly at random from the grammar starting at strings prefixed with s_m (which is certainly possible given an algorithm that samples uniformly from the entire regular/context-free language, as one can derive a language which is also regular/context-free, but starts with strings from the given prefix). Denote this list with $B := \{b_1, \ldots, b_S\}$.
2. Group the samples B into bins C_a of samples with the same first token $a \in \Sigma$, i.e. $C_a = \{b \in B : b = a??\cdots?\}$, where ? stands for any token in the alphabet Σ.
3. Calculate the total of the probabilities of each bin C_a where each element is weighted with respect to the future probabilities given in list W_m, which yields a distribution $D = (d_a)_{a\in\Sigma}$.

It is straightforward to write the unitary $\mathbf{W}'$ that then takes a state $|00\rangle \in \mathcal{H}_F \otimes \mathbb{C}^\Sigma$—the first register for storing a number in F, and the second for storing a letter—and a list of such weights D to a weighted superposition $\mathbf{W}'(D)|0\rangle = \sum_{a\in\Sigma}\sqrt{d_a}|d_a\rangle|a\rangle$ (where for the sake of simplicity we drop the scratch space register that is certainly required). Furthermore, we need a controlled unitary $\mathbf{Q}$ that, given some state $|h\rangle|a\rangle$ where $h = h(a)$ in some specified fashion—which we can demand the $\mathbf{V}_i^a$ produce—uncomputes a and d_a from the second register, i.e. $\mathbf{Q}|h\rangle|d_a\rangle|a\rangle = |h\rangle|00\rangle$.

Together with the sequence of parser unitaries $\mathbf{U}_i'$, the overall quantum circuit $\mathbf{U}_\mu$—depicted in Fig. 3—can then be constructed as follows:

For a partial string $s_1 s_2 \cdots s_i$ of length i, we denote the set of all strings in the grammar prefixed with letters of s with $\mathcal{A}(s_1 \ldots s_i)$. At every step i in the algorithm we sample the expectation value of a future hypothesis continuing with some token a, weighted by their individual likelihood p_{ij}. The sampling procedure then yields an empirical distribution $(d_a)_{a\in\Sigma}$, which we denote with

$$d_a = f_*^{si}(a) = \sum_{j=1}^{S} \chi\left[b_j \in \mathcal{A}(s_1 \ldots s_i a)\right] p(b_j) \Big/ \sum_{j=1}^{S} p(b_j), \tag{10}$$

where the S sampled hypothesis is given in list $B = \{b_1, \ldots, b_S\}$ with individual letters $b_j = b_{j,1} \cdots b_{j,n}$). As usual, $\chi[\cdot]$ denotes the indicator function, and

$$p(b_j) := \prod_{k=1}^{n} p_{k,b_{jk}}.$$

Our goal is to show that the algorithm reproduces the desired weight distribution given in Eq. (3), i.e.

$$\Pr(s) = \prod_{i=0}^{n-1} \Pr(s_{i+1}|s_1 \ldots s_i)$$

where

$$\Pr(s_{i+1}|s_1 \ldots s_i) = \frac{\sum_{x\in\mathcal{A}(s_1\ldots s_i)} p(x)}{\sum_{x\in\mathcal{A}(s_1\ldots s_{i+1})} p(x)}$$

To estimate the total probability distribution to error ϵ in total variation distance, it suffices to approximate each conditional distribution to error ϵ/n, and thus we must show

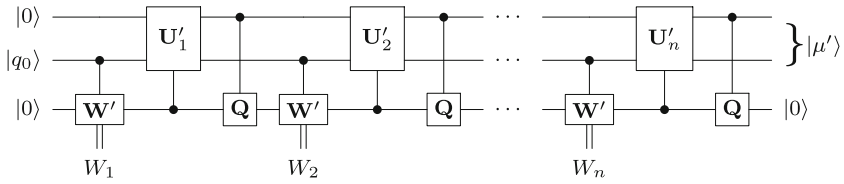


Fig. 3 Quantum algorithm to sample from a language according to weights W_i, constructed in Theorem 3. $|q_0\rangle$ is the automaton's initial internal state

how many samples S are required for d_a to be a good estimator for $\Pr(a|s_1 \dots s_i)$.

First note that $f^{si}(a) = u^{si}(a)/v^{si}$ for

$$u^{si}(a) := \frac{1}{S}\sum_{j=1}^{S} \chi\left[b_j \in \mathcal{A}(s_1 \dots s_i a)\right] p(b_j) \quad \text{and}$$

$$v^{si} := \frac{1}{S}\sum_{j=1}^{S} p(b_j) = \sum_{a\in\Sigma} u^{si}(a).$$

It is straightforward to calculate that

$$\mathbb{E}(u^{si}(a)) = \frac{1}{|\mathcal{A}(s_1\dots s_i)|}\sum_{x\in\mathcal{A}(s_1\dots s_i a)} p(x) \quad \text{and}$$

$$\mathbb{E}(v^{si}) = \frac{1}{|\mathcal{A}(s_1\dots s_i)|}\sum_{x\in\mathcal{A}(s_1\dots s_i)} p(x)$$

and so $\mathbb{E}(u^{si}(a))/\mathbb{E}(v^{si}) = \Pr(a|s_1 \dots s_i)$, the value we are trying to estimate.

Therefore it suffices to take enough samples S such that the $u^{si}(a)$ are close to their mean in relative error (and thus v^{si} is also close in relative error, since $v^{si} = \sum_a u^{si}(a)$).

Noting that $u^{si}(a) = \frac{1}{S}\sum_{j=1}^{S} Y_j$ for i.i.d. random variables Y_j, we have that $\mathrm{Var}(u^{si}(a)) = \frac{1}{S}\mathrm{Var}(Y)$. Therefore by Chebyshev's inequality, to get a ϵ/n relative error approximation requires the number of samples S to be at least

$$S \geq \frac{\mathrm{Var}(Y)}{\mathbb{E}(Y)^2(\epsilon/n)^2}.$$

By assumption $\mathrm{Var}(Y)/\mathbb{E}(Y)^2 \leq \kappa$, and so the total number of uses of the sampler over all n steps of the algorithm is $O(\kappa n^3/\epsilon^2)$ as claimed. □

We note that variants of this sampling algorithm are certainly possible: a naïve approach would be to just sample from the product-of-powerlaws distribution and postselect on the resulting strings being in the grammar; the performance of this will then depend on the number of strings in the grammar vs. the number of all possible strings. Another method could be to execute the uniform sampler in superposition, and perform amplitude amplification on the resulting quantum state to reintroduce the power-law bias. The number of amplification rounds will again depend on the distribution of the strings in the grammar.

Appendix 4. MOST LIKELY PARSE: full query bound

In this section we rigorously prove the integral runtime expression in Lemma 10.

As a reminder, the type of integral from Eq. (6) we are interested in takes the form

$$M(R,k_1,k_2,c,n) := \frac{1}{h_R^n(k_1)}\iiint_1^R \frac{\chi(r_1\cdots r_n \leq c)}{(r_1\cdots r_n)^{k_2}}\mathrm{d}r_1\cdots\mathrm{d}r_n,$$

where k_1 is not necessarily equal to k_2, and typically $c = (R/h_R(k_1))^{n/k_1}$. Here, $\chi(\cdot)$ denotes the characteristic function of a set, i.e. it takes the value 1 where the premise is true, and 0 otherwise. It is possible to integrate Eq. (6) numerically for small n; however, due to the high dimensionality and the flat tail, convergence suffers drastically already for $n > 6$. Similarly, evaluating the integral with a computer algebra system takes significant time for larger n and produces ever growing expressions that are hard to handle, as the reader is welcome to verify. To address this problem, we derive the closed-form expression from Lemma 3:

Lemma 4 *For $k \neq 1$,* Eq. (6) *becomes*

$$M(R,k_1,k_2,c,n) = \frac{(-1)^n}{k'^n h_R^n(k_1)}\sum_{j=0}^{\min\{n,\lfloor c'/a'\rfloor\}}\binom{n}{j} \times\left(\mathrm{e}^{a'k'j} - \mathrm{e}^{-c'k'}\sum_{l=0}^{n-1}\frac{(a'k'j-c'k')^l}{l!}\right),$$

where $k' = 1-k_2$, $c' = \log c$, $a' = \log R$.

Proof As a first step, we perform a log substitution $z_i = \log r_i$, $\mathrm{e}^{z_i}\mathrm{d}z_i = \mathrm{d}r_i$ which yields

$$M(R,k_1,k_2,c,n) = \frac{1}{h_R^n(k_1)}\iiint_0^{\log R}\mathrm{e}^{(1-k_2)(z_1+\dots+z_n)}\chi \times(z_1+\dots+z_n\leq\log c)\mathrm{d}z_1\cdots\mathrm{d}z_n.$$

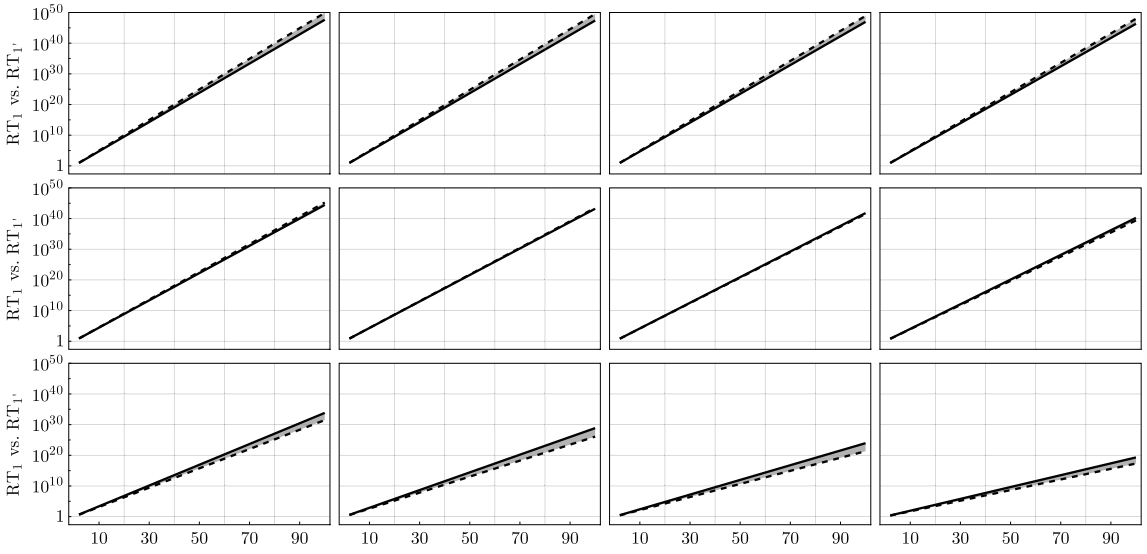


Fig. 4 Expected runtime $\mathrm{RT}_1(R,k,n)$ as evaluated for $R = 10$ and various k (top row: $k \in \{0.2, 0.4, 0.6, 0.8\}$, middle row: $k \in \{1.2, 1.4, 1.6, 1.8\}$, bottom row: $k \in \{2.5, 3, 3.5, 4\}$, always from left to right), vs. the same parameters used for $\mathrm{RT}_{1'}(R,k,n)$ (dashed line), where the discrete probabilities from the power law are approximated with a continuous Pareto distribution. On the x-axis is the length of the input sequence n

The characteristic function is now supported on a rescaled unit simplex, and writing $\bar{z} := \sum_i z_i$ we can take its Fourier transform

$$\begin{aligned}\mathcal{F}_t\chi(\bar{z}\le c') &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}}\chi(\bar{z}\le c')\mathrm{e}^{\mathbb{i}\bar{z}t}\mathrm{d}\bar{z}\\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c'}\mathrm{e}^{\mathbb{i}\bar{z}t}\mathrm{d}\bar{z}\\ &= \frac{\pi}{2}\delta(t)+\frac{\mathrm{e}^{\mathbb{i}c't}}{\sqrt{2\pi}\mathbb{i}t}\\ &=: \tilde{\chi}_{c'}(t).\end{aligned}$$

We of course have $\mathcal{F}_{\bar{z}}^{-1}\mathcal{F}_t\chi \equiv \chi$. Then

$$\begin{aligned}&h_R^n(k_1)M(R,k_1,k_2,c,n)\\ &= \iiint_0^{a'}\mathrm{e}^{k'\bar{z}}\chi(\bar{z}\le c')\mathrm{d}z_1\cdots\mathrm{d}z_n\\ &= \iiint_0^{a'}\mathrm{e}^{k'\bar{z}}\int_{\mathbb{R}}\mathrm{e}^{-\mathbb{i}t\bar{z}}\frac{\tilde{\chi}_{c'}(t)}{\sqrt{2\pi}}\mathrm{d}t\ \mathrm{d}z_1\cdots\mathrm{d}z_n\\ &\overset{*}{=} \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}}\tilde{\chi}_{c'}(t)\left(\prod_{l=1}^n\int_0^{a'}\mathrm{e}^{(k'-\mathbb{i}t)z_l}\mathrm{d}z_l\right)\mathrm{d}t\\ &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}}\tilde{\chi}_{c'}(t)\left(\frac{\mathrm{e}^{a'(k'-\mathbb{i}t)}-1}{k'-\mathbb{i}t}\right)^n\mathrm{d}t\\ &= \frac{(-1)^n}{\sqrt{2\pi}}\int_{\mathbb{R}}\sum_{j=0}^n\binom{n}{j}(-1)^j\frac{1}{(k'-\mathbb{i}t)^n}\mathrm{e}^{a'(k'-\mathbb{i}t)j}\tilde{\chi}_{c'}(t)\mathrm{d}t\\ &= \frac{1}{\sqrt{2\pi}}\sum_{j=0}^n\binom{n}{j}\mathrm{e}^{a'k'j}\frac{(-1)^j}{\mathbb{i}^n}\underbrace{\int_{\mathbb{R}}\frac{\mathrm{e}^{-\mathbb{i}a'jt}}{(t+\mathbb{i}k')^n}\tilde{\chi}_{c'}(t)\mathrm{d}t}_{=:J_n}.\end{aligned}\tag{11}$$

In the step marked with $*$, we applied Fubini's theorem, for which we implicitly assumed a smooth limiting argument for the step function. To evaluate the integral J_n, we observe that the denominator has a root of order n at

$$t_0 = -\mathbb{i}k' = \begin{cases}+\mathbb{i}|k'| & k>1 \Leftrightarrow k'<0\\ -\mathbb{i}|k'| & k<1 \Leftrightarrow k'>0\\ 0 & k=1\Leftrightarrow k'=0.\end{cases}$$

We further expand the Fourier-transformed characteristic function—and again glossing over the details of Fubini's theorem to swap the integration order—to obtain

$$J_n = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c'}\int_{\mathbb{R}}\frac{\mathrm{e}^{\mathbb{i}t(x-ja')}}{(t+\mathbb{i}k')^n}\mathrm{d}t\ \mathrm{d}x.\tag{12}$$

We handle the integrand's three pole cases separately.

k > 1. We have $k' < 0$ and an order n pole at $\mathbb{i}|k'|$; the integrand $g(t) := \mathrm{e}^{\mathbb{i}t(x-ja')}/(t+\mathbb{i}k')^n$ is holomorphic in the lower half plane. The exponent of the exponential, $x - ja'$, assumes signs

$$x-ja'\begin{cases}>0 & x>ja'\\ <0 & x<ja'\\ =0 & x=ja'.\end{cases}$$

In the latter case, the integral (over t) evaluates to zero.

In the middle case, for $t = -\mathbb{i}s$ we have $\exp(\mathbb{i}(-\mathbb{i})s(x-ja')) = \exp(s(x-ja')) \longrightarrow 0$ as $s \longrightarrow \infty$ as $s \longrightarrow \infty$; by Jordan's lemma we can thus write

$$\int_{\mathbb{R}} g(t)\mathrm{d}t = \lim_{r\to\infty}\oint_{\gamma_1(r)} g(t)\mathrm{d}t = 0,$$

where $\gamma_1(r)$ contains the real interval $[-r, r]$ and a half circle connecting the end points in the lower half complex plane.

In the first case, for $t = \mathbb{i}s$, we have $\exp(\mathbb{i}^2 s(x - ja')) = \exp(-s(x - ja')) \longrightarrow 0$ as $s \longrightarrow \infty$; however now the corresponding upper half plane loop encircles the pole of $g(x)$. We apply the residue theorem for a flipped path $\gamma_2(r) = -\gamma_1(r)$:

$$\begin{aligned}\int_{\mathbb{R}} \frac{\mathrm{e}^{\mathbb{i}t(x-ja')}}{(t+\mathbb{i}k')^n}\mathrm{d}t &= \lim_{r\to\infty}\oint_{\gamma_2(r)} \frac{\mathrm{e}^{\mathbb{i}t(x-ja')}}{(t+\mathbb{i}k')^n}\mathrm{d}t \\ &= 2\pi\,\mathbb{i}\mathrm{Res}_t(g, t_0) \\ &= \frac{2\pi\,\mathbb{i}}{(n-1)!}\lim_{t\to t_0}\frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}}((t-t_0)^n g(t)) \\ &= \frac{2\pi\,\mathbb{i}}{(n-1)!}\lim_{t\to t_0}(\mathbb{i}(x-ja'))^{n-1}\mathrm{e}^{\mathbb{i}t(x-ja')} \\ &= \frac{2\pi\,\mathbb{i}^n}{(n-1)!}(x-ja')^{n-1}\mathrm{e}^{k'(x-ja')}.\end{aligned}$$

For the case $x - ja' < 0$ we are left to perform the outer integration in Eq. (12). If $c' \leq ja'$ we necessarily have $x \leq ja'$ and $J_n = 0$. For the case $c' > ja'$ we have

$$\begin{aligned}&\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c'}\frac{2\pi\,\mathbb{i}^n}{(n-1)!}(x-ja')^{n-1}\mathrm{e}^{k'(x-ja')}\mathrm{d}x \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(n-1)!}\int_0^{c'-ja'} y^{n-1}\mathrm{e}^{k'y}\mathrm{d}y \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(n-1)!}\frac{1}{(-k')^n}(\Gamma(n)-\Gamma(n, a'k'j-c'k')) \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(-k')^n}\left(1-\frac{\Gamma(n, a'k'j-c'k')}{\Gamma(n)}\right),\end{aligned}$$

Where $\Gamma(n, \cdot)$ is the lower incomplete gamma function. Putting it all together, we get

$$\begin{aligned}J_n &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c'}\int_{\mathbb{R}}\frac{\mathrm{e}^{\mathbb{i}t(x-ja')}}{(t+\mathbb{i}k')^n}\mathrm{d}t\;\mathrm{d}x \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(-k')^n}\begin{cases}0 & c' \leq ja' \\ 1-\frac{\Gamma(n,a'k'j-c'k')}{\Gamma(n)} & \text{otherwise.}\end{cases}\end{aligned}$$

Finally, we insert the last expression back into Eq. (11), and obtain

$$\begin{aligned}M(R, k_1, k_2, c, n) &= \frac{1}{h_R^n(k_1)}\frac{1}{\sqrt{2\pi}}\sum_{j=0}^{n}\binom{n}{j}\mathrm{e}^{a'k'j}\frac{(-1)^j}{\mathbb{i}^n}J_n \\ &= \frac{(-1)^n}{k'^n h_R^n(k_1)}\sum_{j=0}^{\min\{n,\lfloor c'/a'\rfloor\}}\binom{n}{j}\mathrm{e}^{a'k'j} \\ &\quad\times\left(1-\frac{\Gamma(n, a'k'j-c'k')}{\Gamma(n)}\right).\end{aligned}$$

The second term in the sum we can further simplify using the identity $\Gamma(n, x)/\Gamma(n) = \mathrm{e}^{-x}\sum_{l=0}^{n-1} x^l/l!$ which holds for integer j, which yields

$$\begin{aligned}M(R, k_1, k_2, c, n) &= \frac{(-1)^n}{k'^n h_R^n(k_1)}\sum_{j=0}^{\min\{n,\lfloor c'/a'\rfloor\}}\binom{n}{j} \\ &\quad\times\left(\mathrm{e}^{a'k'j}-\mathrm{e}^{-c'k'}\sum_{l=0}^{n-1}\frac{(a'k'j-c'k')^l}{l!}\right).\end{aligned}$$

k < 1. We have $k' > 0$ and the order n pole of Eq. (12) lies at $-\mathbb{i}|k'|$. The integrand $g(t) = \mathrm{e}^{\mathbb{i}t(x-ja')}/(t+\mathbb{i}k')^n$ is holomorphic in the upper half plane; and analogous to before, this time when $x - ja' > 0$, we have

$$\int_{\mathbb{R}} g(t)\mathrm{d}t = \lim_{r\to\infty}\oint_{\gamma_2(r)} g(t)\mathrm{d}t = 0.$$

In the opposite case we can again apply the residue theorem and obtain

$$\int_{\mathbb{R}} g(t)\mathrm{d}t \overset{*}{=} -2\pi\,\mathbb{i}\mathrm{Res}_t(g, t_0) = -\frac{2\pi\,\mathbb{i}^n}{(n-1)!}(x-ja')^{n-1}\mathrm{e}^{k'(x-ja')},$$

where the negative sign in step $*$ stems from the clockwise orientation of the contour γ_2. The outer integration in Eq. (12) is now

$$\begin{aligned}J_n &= -\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c'}\frac{2\pi\,\mathbb{i}^n}{(n-1)!}\begin{cases}(x-ja')^{n-1}\mathrm{e}^{k'(x-ja')} & x < ja' \\ 0 & \text{otherwise}\end{cases}\mathrm{d}x \\ &= -\frac{\sqrt{2\pi}\,\mathbb{i}^n}{(n-1)!}\int_{-\infty}^{\min\{c', ja'\}}(x-ja')^{n-1}\mathrm{e}^{k'(x-ja')}\mathrm{d}x \\ &= -\frac{\sqrt{2\pi}\,\mathbb{i}^n}{(n-1)!}\int_{-\infty}^{\min\{c'-ja, 0\}} y^{n-1}\mathrm{e}^{k'y}\mathrm{d}y \\ &= -\frac{\sqrt{2\pi}\,\mathbb{i}^n}{(n-1)!}(-1)^{n+1}k'^{-n}\Gamma(n, -k'\min\{c'-ja', 0\}) \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(-k')^n}\frac{\Gamma(n, \min\{a'k'j-c'k', 0\})}{\Gamma(n)} \\ &= \frac{\sqrt{2\pi}\,\mathbb{i}^n}{(-k')^n}\begin{cases}1 & c' \geq ja' \\ \frac{\Gamma(n,a'k'j-c'k')}{\Gamma(n)} & \text{otherwise.}\end{cases}\end{aligned}$$

Inserting the expression back into Eq. (11) we obtain

$$\begin{aligned}M(R, k_1, k_2, c, n) &= \frac{1}{h_R^n(k_1)}\frac{1}{\sqrt{2\pi}}\sum_{j=0}^{n}\binom{n}{j}\mathrm{e}^{a'k'j}\frac{(-1)^j}{\mathbb{i}^n}J_n \\ &= \frac{(-1)^n}{k'^n h_R^n(k_1)}\left[\sum_{j=0}^{\min\{n,\lfloor c'/a'\rfloor\}}\binom{n}{j}(-1)^j\mathrm{e}^{a'k'j} + \right. \\ &\quad\left.\sum_{j=\lfloor c'/a'\rfloor+1}^{n}\sum_{l=0}^{n-1}\binom{n}{j}(-1)^j\mathrm{e}^{c'k'}\frac{(a'k'j-c'k')^l}{l!}\right].\end{aligned}$$

To reduce the last sum to the previous expression, we note that

$$\sum_{j=0}^{n}\sum_{l=0}^{n-1}\binom{n}{j}(-1)^j\frac{(xj-y)^l}{l!}$$
$$=\sum_{l=0}^{n-1}\frac{x^l}{l!}\sum_{j=0}^{n}\binom{n}{j}(-1)^j\sum_{m=0}^{l}\binom{l}{m}j^m\left(-\frac{y}{x}\right)^{l-m}$$
$$=\sum_{l=0}^{n-1}\frac{x^l}{l!}\sum_{m=0}^{l}\binom{l}{m}\left(-\frac{y}{x}\right)^{l-m}\underbrace{\sum_{j=0}^{n}\binom{n}{j}(-1)^j j^m}_{=(-1)^n n!S_m^{(n)}},$$

where $S_m^{(n)}$ is the Stirling number of the second kind, which denotes the number of ways to partition a set of size m into n non-empty subsets. Since $m \le l \le n-1$, $S_m^{(n)} \equiv 0$ here, and thus

$$\sum_{l=0}^{n-1}\sum_{j=\lfloor c'/a'\rfloor+1}^{n}\binom{n}{j}(-1)^j\frac{(a'k'j-c'k')^l}{l!}$$
$$=-\sum_{l=0}^{n-1}\sum_{j=0}^{\min\{n,\lfloor c'/a'\rfloor\}}\binom{n}{j}(-1)^j\frac{(a'k'j-c'k')^l}{l!}.$$

The claim follows. □

We leave the $k=1$ case as an exercise to the reader.

With Lemma 3, we can now evaluate the terms in Eq. (5) efficiently. The first term is

$$\mathrm{rt}=\sqrt{h_R^n(k)}M\left[R,k,\frac{k}{2},\left(\frac{R}{h_R(k)}\right)^{\frac{n}{k}},n\right], \tag{13}$$

and the second

$$\mathrm{rt}'=1-M\left[R,k,k,\left(\frac{R}{h_R(k)}\right)^{\frac{n}{k}},n\right]. \tag{14}$$

Of interest is whether taking this full expectation value and splitting it to fall back to Grover search whenever the probability dips below $1/R^n$ yields a significant improvement of the runtime bound. We found this to not be the case, as Fig. 5 demonstrates; while for smaller n there is a significant improvement, as n grows the ratio $\mathrm{rt}/\mathrm{RT}_1 \longrightarrow 1$ exponentially fast.

Appendix 5. Postselected product of powerlaws

In this appendix we answer the open question left in Section 4. The setup here is as follows. Let $S > 1$, and X be distributed according to a product of distributions with pdf

$$p(r_1,\ldots,r_n) := \frac{1}{H_S(k)^n}\frac{1}{(r_1\cdots r_n)^k}$$

as in Eq. (4), i.e. where every factor is a power law distribution over S elements. If we remove a random subset of the elements such that R^n (for some $R < S$) elements are left over, is the resulting probability distribution a product-of-powerlaws, where every factor is over R elements?

In the continuous case this can be seen as follows. If $X \sim \mathrm{Pareto}(k,S)$ with pdf p as defined in Section 4, then removing a random subset of elements on the interval $[1, S+1]$ is equivalent to taking a random characteristic function χ_1 over it, with $\int_{[1,S+1]}\chi(r)\mathrm{d}r = R$, and defining X' with pdf $Sp(r)\chi(r)/R$. We define the postselected random variable Y over $[1,R]$ by relabelling the points in $\mathrm{supp}\chi$ by values in $[1,R]$ in an order-preserving fashion.

Similarly, if X_n is a product of n iid Pareto random variables with pdf p_n, then postselection means taking a random characteristic function χ_n on $[1,S]^n$ with

$$\int_{[1,S+1]^n}\chi_n(r_1,\ldots,r_n)\mathrm{d}r_1\cdots\mathrm{d}r_n = R^n.$$

We claim that the resulting random variable X_n' with pdf $S^n p_n(r)\chi_n(r)/R^n$ then factors into a product distribution. This holds because χ_n has the property that for all $\epsilon > 0$ there exists a bijection f such that for almost all $(x_1,\ldots,x_n) \in \mathrm{supp}\chi_n$, there exists

$$(y_1,\ldots,y_n) = f(x_1,\ldots,x_n) \in (\mathrm{supp}\chi_1)^n$$
$$\text{s.t.}\quad \sum_{i=1}^{n}|x_i-y_i| < \epsilon,$$

for some characteristic function χ_1 defined on $[1,S]$. We refer to this property as χ_n being 'product'.

We now prove this claim by induction on n. For $n=1$, X and χ_1 are already product, so there is nothing to show. Assume the hypothesis holds for χ_n which can be factored

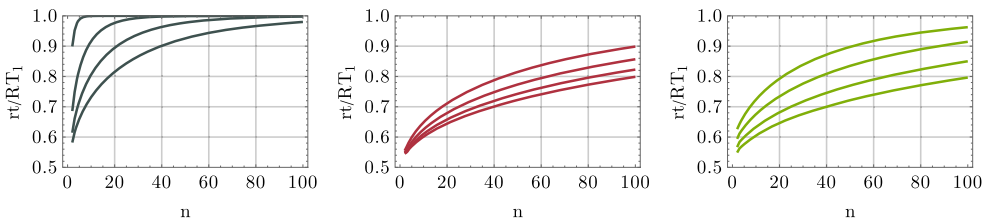


Fig. 5 Ratios of $\mathrm{rt}/\mathrm{RT}_1$ for various power law exponents k. left: {0.2, 0.4, 0.6, 0.8} from top to bottom, middle: {1.2, 1.4, 1.6, 1.8} from top to bottom, right: {2.5, 3.0, 3.5, 4.0} from bottom to top. In all cases the runtime ratios approach 1 exponentially fast with growing n

into a product χ_1^n for some χ_1. Take a random characteristic function χ_{n+1} over $n+1$ dimensions. Let $\epsilon > 0$. As $\mathbb{R}$ is uncountable, we take a δ-net over the interval $I := [1, S+1]$ for some small $\epsilon \gg \delta > 0$, which we will denote with I_δ; each $x \in I$ then has a corresponding $x' \in I_\delta$ that satisfies $|x - x'| < \delta$. In particular, I_δ is countable. In a similar fashion, for χ_{n+1} we consider its discretised variant over I_δ^n as χ'_{n+1}.

So let $(x_1, \ldots, x_n, x_{n+1}) \in \text{supp}\chi'_{n+1}$, and analogously define the discretised characteristic functions χ'_n and χ'_1. A counting argument shows that within each ϵ-bin (defined over I and extended over to I_δ^n accordingly), we can map $(x_1, \ldots, x_{n+1})$ to their closest corresponding point $(y_1, \ldots, y_n, z_1) \in \chi'_n \times \chi'_1$—or if that point was previously chosen its next- and next-to-next-closest one etc., while staying within ϵ distance for each original coordinate for the majority of the points.

A limiting argument $\epsilon \longrightarrow 0$ shows that this map can be constructed for almost all points. This concludes the induction.

The question that remains is what distribution Y follows. Despite scale invariance of Pareto distributions, the resulting pdf for a surviving fraction λ of the original points looks like $\tilde{p}(r) = p(1 + (x-1)\lambda)$, which is itself not a Pareto distribution. Yet, since we actually work with a power law distribution, we already answered in Section 4 what this resulting sample distribution over R looks like: it can be well-approximated by a power law with a slightly worse falloff $k' < k$ that itself can be estimated numerically in a straightforward fashion. The smaller exponent should also account for any approximation errors made by the continuous variable analysis demonstrated in this section.

Appendix 6. Beam search variants

Continuing from Section 5 from the main text, the relevant questions to ask here is what choice of p_0 will

1. only require a constant—or logarithmic—number of rounds of amplitude amplification,
2. retain a large number of hyptheses, and
3. improve runtime for the post-amplified QUANTUM-SEARCHDECODE variant.

We address all these questions in the next sections.

6.1 Constant post-amplification

In light of simplicity, we will take RT_1 as an upper runtime bound to the full expected number of rounds, RT_2; as we amplify away all paths with weights below the cutoff we never expect to find an element therein—meaning we can drop the fallback to Grover search in our analysis, and treat the search as if the advice state was purely on those paths with weight $\geq p_0$.

We first address the question for which choice of p_0 the cumulative leftover probability $M(R, k, k, p_0, n)$ can be lower-bounded by a quantity independent of n, which means we have to perform only a *constant* number of amplitude amplification rounds on the advice state. In order to do so, we solve the implicit inequality

$$\text{minimize } f_{\text{split}} \text{ subject to } M\left[R, k, k, \underbrace{\left(\frac{R}{h_R(k)}\right)^{\frac{n}{k} f_{\text{split}}}}_{=p_0}, n\right] \geq C_0. \tag{15}$$

As M is monotonically decreasing for a decreasing splitting exponent f_{split}, and since M can be computed in $\text{O}(n^2)$ many arithmetic operations, we can perform the minimisation efficiently. For a choice of $C_0 = 1/4$ (which implies a single amplitude amplification round) and $C_0 = 1/100$ (ten rounds of amplification) we plot f_{split} in Fig. 6. As can be seen, f_{split} tends towards a limiting value $\in (0, 1)$ for $n \longrightarrow \infty$.

The next step in our analysis is to take the modified splitting exponent f_{split} and count how many hypotheses N_{hyp} remain to be searched over; this is important because it is not clear a priori how many paths we can still search over, and if that quantity is low—or even tends towards zero—then we retained too few elements. Our hope is of course that in contrast to beam search, where generally the beam's width, i.e. the number of hypotheses retained at any point in time, is capped at some possibly large but constant value, we have a growing number of hypotheses to search over.

In order to count this number of hypotheses given a cutoff probability p_0, we can evaluate $M(R, k, k, p_0, n)$ in the limit of the power law exponent $k \longrightarrow 0$, and finally multiply $h_R^n(k_1)$ in Eq. (6) to make the integral *count* instead of calculating a cumulative density. We again choose a series of values for R, k and C_0 and plot the results in Fig. 7. While the number of leftover hypotheses is indeed reduced drastically as compared to performing a full search over R^n elements, it is still growing exponentially with n, which results in a significant number of hypotheses to search over, many more than possible in the classical setting.

As a last step, we want to analyse the modified runtime given the changed probability cutoff, which corresponds to evaluating the integral $M(R, k, k/2, p_0, n)$ with the p_0 derived from the optimisation Eq. (15). The results are collected in Fig. 8. As one can verify, the runtime does remain asymptotically exponential in the sequence length n; however the base of the exponential is reduced accordingly.

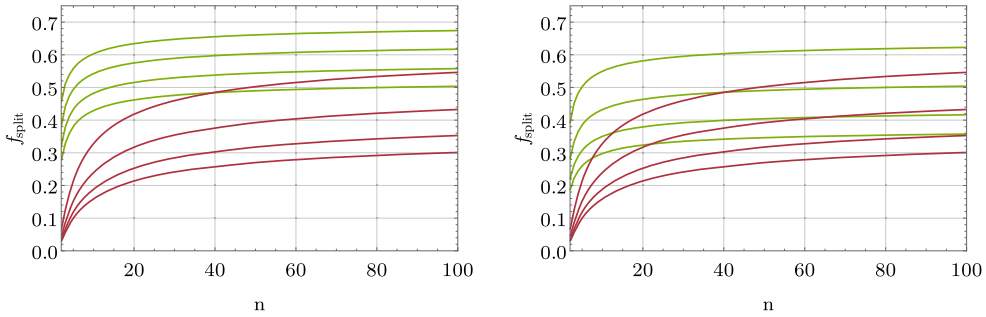


Fig. 6 Minimized value of the splitting exponent f_{split} as defined in Eq. (15). Plotted are the values for $R = 6$ (left) and $R = 24$ (right), as well as $C_0 = 1/4$ (green, upper family of lines) which implies exactly one extra round of amplitude amplification, and $C_0 = 1/100$ (red, lower family of lines) which implies ten extra rounds of amplification. The power law exponents chosen are $k \in \{1.5, 2.0, 2.5, 3.0\}$ (bottom to top, respectively)

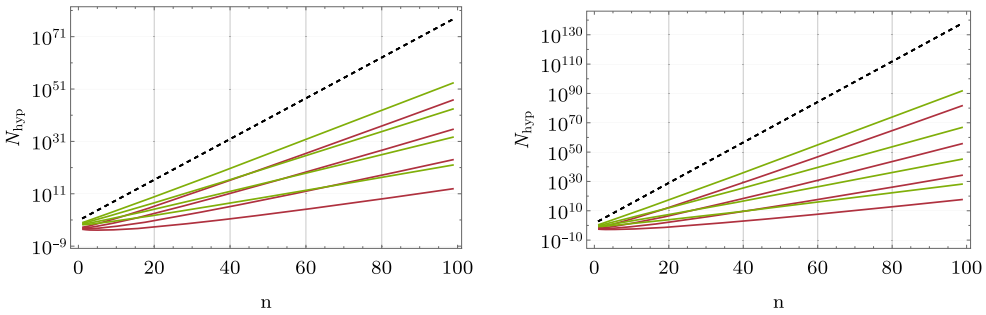


Fig. 7 Number of hypotheses N_{hyp} left for a specific choice of splitting exponent f_{split} to retain $C_0 > 1/4$ (green, one extra round of amplification) and $C_0 > 1/100$ (red, ten extra rounds of amplification) total probability weight for the hypotheses. The value of f_{split} is obtained numerically from Eq. (15) (cf. Fig. 6). Plotted is the case $R = 6$ (left) and $R = 24$ (right), and $k \in \{1.5, 2.0, 2.5, 3.0\}$ (from top to bottom in each plot and each color, respectively). The dashed line is the total number of possible hypotheses R^n as reference

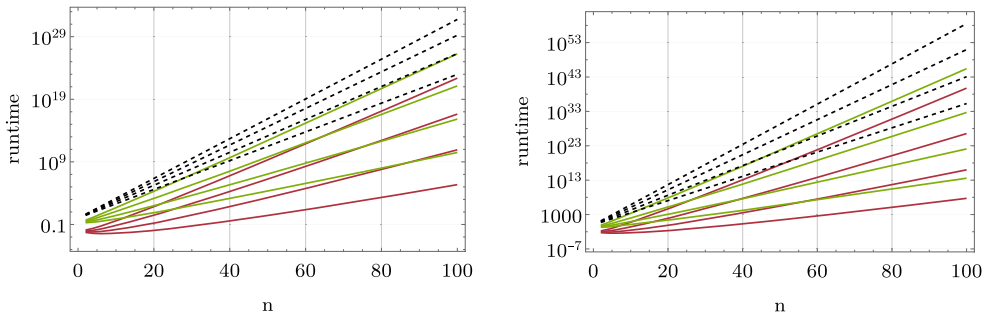


Fig. 8 Runtime when post-amplifying to retain only a fraction $C_0 \geq 1/4$ of weight (green, one extra round of amplification) or $C_0 \geq 1/100$ (red, ten extra rounds of amplification) on the hypotheses. The value of f_{split} is obtained numerically from Eq. (15) (cf. Fig. 6). Plotted is the case $R = 6$ (left) and $R = 24$ (right), and $k \in \{1.5, 2.0, 2.5, 3.0\}$ (from top to bottom in each plot and for each color, respectively). The dashed line is the full search runtime $\text{RT}_1(R, k, n)$ from Lemma 2 as reference

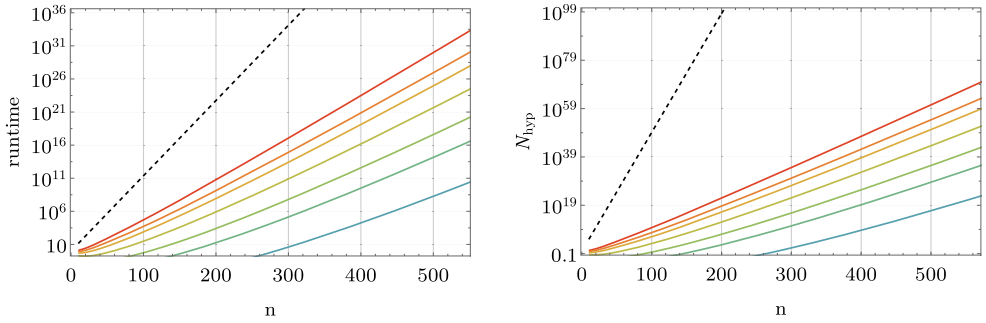


Fig. 9 Number of iterations (Eq. (18)) and number of hypotheses (Eq. (17)) of quantum beam search decoding the output of Mozilla's *DeepSpeech* LSTM with a grammar, assuming an average branching ratio of $R = 3$, a token power law distribution with exponent $k = 2.91$, and post-amplification of the quantum search decoder with a retained fraction of hypotheses $C_0 = C_0(n) \in \{n^{-1/2}, n^{-2/3}, n^{-1}, n^{-3/2}, n^{-2}, n^{-3}\}$ as defined in Eq. (16), which is plotted in rainbow colors from red to blue, top to bottom. The dashed line is the full quantum search runtime and number of hypotheses from Eq. (13)

6.2 Non-constant post-amplification

The analysis of Appendix 6.1 can of course be repeated for a non-constant f_{split}; however, one has to be aware that these extra amplitude amplification rounds factor into the overall runtime. For a retained fraction $g(n)$ of the total probability weight, the optimisation thus reads

$$\text{minimize } p_0 \text{ subject to} \quad M(R,k,k,p_0,n) \geq g(n) \tag{16}$$

$$\text{which retains } \lim_{k\to 0} \quad M(R,k,k,p_0,n) \text{ hypotheses,} \tag{17}$$

$$\text{and has runtime bound} \quad g(n)^{-1/2} M(R,k,k/2,p_0,n). \tag{18}$$

We take the power law exponent derived from Mozilla's DeepSpeech neural network, $k = 3.03$ (cf. Sec. 5.2, supplementary material), and derive runtime bounds for decoding its output with a parser under the assumption that, on average, we take $R = 3$ branches in the parsing tree at every time step. As discussed in Section 4, the sampling distribution over three elements only yields a slightly lower exponent of $k = 2.91$.

As an example we consider an input sequence of length 500; with the above parameters and a splitting exponential $f_{\text{split}} = n^{-1/2}$ (resp. $= n^{-3}$) we can search over $N_{\text{hyp}} \approx 10^{60}$ (resp. $\approx 10^{18}$) hypotheses, with a runtime $\approx 10^{30}$ (resp. $\approx 10^{9}$). Similarly, when capping the beam width at $N_{\text{hyp}} \leq 10^6$, we asymptotically require $\approx 10^3$ iterations of the beam search decoder (which includes the post-amplification rounds); for shorter sequences, a super-Grover speedup as present in full QUANTUMSEARCHDECODE is achieved (Fig. 9).

Appendix 7. Further proof details

For Lemma 5, a more detailed proof is given as follows.

Lemma 5 By Th. 1 (Buhrman et al. 2001), we have that any non-reversible computation requiring time T and space S can be simulated reversibly in time $T' = 3^k 2^{O(T/2^k)}$ and space $S' = (1 + O(k))S$, for a $0 \leq k \leq \log_2 T$ chosen arbitrarily. Choose $k = \log_2 T$, then $S' = (1 + O(\log_2 T))S$, and $T' = O(T^{\log_2 3})$. Now translate this reversible probabilistic classical circuit into a quantum circuit—e.g. using the Solovay-Kitaev theorem (Nielsen and Chuang 2010), which incurs an at most logarithmic runtime overhead. □

Appendix 8. Rank of letter likelihood for Mozilla's DeepSpeech

DeepSpeech processes mel-frequency cepstral coefficients extracted from a sliding window of 25 ms, with a stride of 20 ms; for each such frame, the LSTM is invoked, and yields a distribution over the letters of the English alphabet "a" to "z", as well as a few special symbols, e.g. "silence". For the specific architecture of the LSTM we refer the reader to the original paper (Hannun et al. 2014). Our hypothesis was that these letter probabilities follow a power-law distribution; our data (shown in Fig. 10) supports this claim.

We want to emphasise that the fact the letters a–z follow Zipf's law with respect to their occurence in English sentences (see, e.g., Egghe 2000; Piantadosi 2014) plays no role in attaining the speedup. In addition to Fig. 10, we verified that when only collecting those output frames of *DeepSpeech* where, say, "t" is the most likely prediction, the distribution over all letters—sorted by rank, i.e. sorted from most to least likely prediction—is already a power-law. This is a feature of the output of the model, and not necessarily a property of the underlying data the model was trained on. In our context this means that the Softmax output layer of the LSTM has to yield a power-law probability distribution.

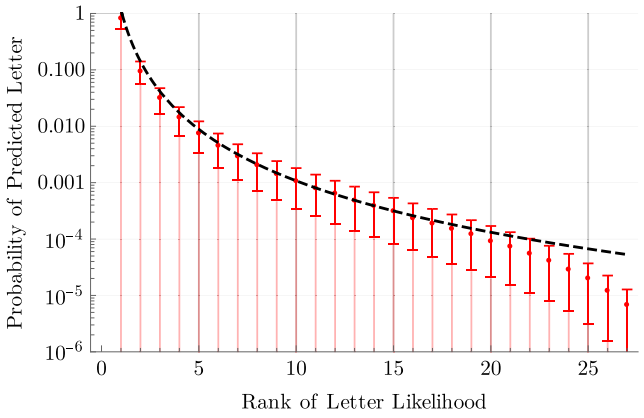


Fig. 10 Log plot of the power law distribution of the output probabilities obtained from Mozilla's *DeepSpeech* voice recognition LSTM on the Mozilla Common Voice verified test dataset for English (Mozilla 2019a), which consists of 3995 audio samples of about 10 s each of spoken test sentences. The dashed line is a fitted power law ar^{-b} with parameters $a = 1.2 \pm 0.1$ and $b = 3.03 \pm 0.03$. We individually process each audio file, and capture the output after the final `Softmax` layer (`logits:0`), but before it is processed further by the greedy connectionist temporal classification (CTC beam search) implemented by *DeepSpeech*

How frequently a given letter is the most likely prediction—which is itself known to be a power-law, as mentioned—is not important.

Acknowledgements J. B. would like to thank the Draper's Research Fellowship at Pembroke College. S. S. would like to thank the Science Education and Research Board (SERB, Govt. of India) and the Cambridge Trust for supporting his PhD through a Cambridge-India Ramanujan scholarship. We are grateful for the useful feedback and the comments we recieved from Jean Maillard, Ted Briscoe, Aram Harrow, Massimiliano Goldwurm, Mark Jerrum, and when presenting this work at IBM Zürich. We further thank Terence Tao for the suggestion to try to take the Fourier transform of the indicator function in Section 4.3.

References

Aaronson S, Grier D, Schaeffer L (2019) A quantum query complexity trichotomy for regular languages. In: IEEE 60th annual symposium on foundations of computer science (FOCS). IEEE, pp 942–965

Ahuja A, Kapoor S (1999) A quantum algorithm for finding the maximum

Al-Rfou R, Choe D, Constant N, Guo M (2019) Character-level language modeling with deeper self-attention. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 3159–3166

Bausch J (2018) Classifying data using near-term quantum devices. Int J Quantum Inf 16(08):1840001

Bausch J (2020) Recurrent quantum neural networks. In: Advances in neural information processing systems, vol 33. 34th Annual conference on neural information processing systems (NeurIPS)

Buckman J, Ballesteros M, Dyer C (2016) Transition-based dependency parsing with heuristic backtracking. In: Proceedings of the 2016 Conference on empirical methods in natural language processing, pp 2313–2318. Stroudsburg, PA, USA. ACL (Association for Computational Linguistics), Association for Computational Linguistics

Berry DW, Childs AM, Cleve R, Kothari R, Somma RD (2014) Exponential improvement in precision for simulating sparse hamiltonians. In: Proceedings of the forty-sixth annual ACM symposium on theory of computing, STOC '14. ACM, New York, pp 283–292

Berry DW, Childs AM, Ostrander A, Wang G (2017) Quantum algorithm for linear differential equations with exponentially improved dependence on precision. Commun Math Phys 356(3):1057–1081

Bernardi O, Giménez O (2012) A linear algorithm for the random sampling from regular languages. Algorithmica 62(1-2):130–145

Bohnet B, McDonald R, Pitler E, Ma J (2016) Generalized transition-based dependency parsing via control parameters. In: Proceedings of the 54th Annual meeting of the association for computational linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Stroudsburg, pp 150–160

Buhrman H, Tromp J, Vitányi P (2001) Time and space bounds for reversible simulation. J Phys A Math 34(35):6821–6830

Babbush R, Wiebe N, McClean J, McClain J, Neven H, Chan GK-L (2018) Low-depth quantum simulation of materials. Phys Rev X 8(1):011044

Chomsky N (1956) Three models for the description of language. IEEE Trans Inform Theory 2(3):113–124

Cox R (2007) Regular expression matching can be simple and fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)

Childs AM, Su Y (2019) Nearly optimal lattice simulation by product formulas. Phys Rev Lett 123(5):050503

Dabrowska E (2008) Questions with long-distance dependencies: A usage-based perspective. Cogn Linguist 19(3)

Dyer C, Ballesteros M, Ling W, Matthews A, Smith NA (2015) Transition-based dependency parsing with stack long short-term memory. In: Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (Volume 1: Long Papers). Association for Computational Linguistics, Stroudsburg, pp 334–343

Denise A (1996) Génération aléatoire uniforme de mots de langages rationnels. Theor Comput Sci 159(1):43–63

Dürr C, Høyer P (1996) A quantum algorithm for finding the minimum in LANL e-print quantph/9607014

Denise A, Roques O, Termier M (2000) Random generation of words of context-free languages according to the frequencies of letters. In: Mathematics and computer science. Basel, Birkhäuser Basel, pp 113–125

Dai Z, Yang Z, Yang Y, Carbonell J, Le QV, Salakhutdinov R (2019) Transformer-XL: attentive language models beyond a fixed-length

context. In: Proceedings of the 57th Annual meeting of the association for computational linguistics
Earley J (1970) An efficient context-free parsing algorithm. Commun ACM 13(2):94–102
Egghe L (2000) The distribution of N-Grams. Scientometrics 47(2):237–252
Fan A, Lewis M, Dauphin Y (2018) Hierarchical neural story generation. In: Proceedings of the 56th annual meeting of the association for computational linguistics
Gilyén A, Arunachalam S, Wiebe N (2019) Optimizing quantum optimization algorithms via faster quantum gradient computation. In: Proceedings of the Thirtieth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 1425–1444
Goldwater S, Griffiths TL, Johnson M (2011) Producing power-law distributions and damping word frequencies with two-stage language models. J Mach Learn Res 12:2335–2382
Gidney C (2018) Halving the cost of quantum addition. Quantum 2:74
Gore V, Jerrum Mark, Kannan S, Sweedyk Z, Mahaney S (1997) A quasi-polynomial-time algorithm for sampling words from a context-free language. Inf Computat 134(1):59–74
Goldwurm M, Palano B, Santini M (2001) On the circuit complexity of random generation problems for regular and context-free languages. In: Ferreira A, Reichel H (eds) STACS 2001. Springer, Berlin, pp 305–316
Graves A (2013) Generating sequences with recurrent neural networks
Holtzman A, Buys J, Du L, Forbes M, Choi Y (2020) The curious case of neural text degeneration. In: International conference on learning representations
Hickey T, Cohen J (1983) Uniform random generation of strings in a context-free language. SIAM J Comput 12(4):645–655
Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY (2014) Deep speech: Scaling up end-to-end speech recognition
Harrow AW, Hassidim A, Lloyd S (2009) Quantum algorithm for linear systems of equations. Phys Rev Lett 103(15):150502
Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to automata theory, languages, and computation, vol 32, 2nd edn.
Jäger G. (2012) Power laws and other heavy-tailed distributions in linguistic typology. Adv Complex Syst 15(03n04):1–21
Khandelwal U, He H, Qi P, Jurafsky D (2018) Sharp nearby, fuzzy far away: how neural language models use context. In: Proceedings of the 56th annual meeting of the association for computational linguistics (Volume 1: Long Papers). Association for Computational Linguistics., Stroudsburg, pp 284–294
Kitaev N, Kaiser L, Levskaya A (2020) Reformer: The efficient transformer. In: International conference on learning representations
Kleene SC (1956) Representation of events in nerve nets and finite automata. In: Automata studies. (AM-34). Princeton University Press, Princeton, pp 3–42
Kulikov I, Miller AH, Cho K, Weston J (2018) Importance of a search strategy in neural dialogue modelling
Kerenidis I, Prakash A (2016) Quantum recommendation systems
Li T, Chakrabarti S, Wu X (2019) Sublinear quantum algorithms for training linear and kernel-based classifiers. In: ICML
Lloyd S (1996) Universal quantum simulators. Science 273(5278):1073–1078
Louden KC (1997) From a Regular Expression to an NFA. Pearson/Addison Wesley, Boston
Lorenz WA, Ponty Y (2013) Non-redundant random generation algorithms for weighted context-free grammars, vol 502
Murray K, Chiang D (2018) Correcting length bias in neural machine translation. In: Proceedings of the third conference on machine translation: research papers. Association for Computational Linguistics, Stroudsburg, pp 212–223
McKenzie B (1997) Generating strings at random from a context free grammar. Technical report, Department of Computer Science, University of Canterbury, Engineering Reports
Minsky ML (1967) Unsolvability of the halting problem. Prentice-Hall, Inc, Upper Saddle River
Montanaro A (2011) Quantum search with advice. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), 6519 LNCS, pp 77–93
Montanaro A (2016) Quantum algorithms: an overview. Npj Quantum Inf 2(1):15023
Montanaro A (2017) Quantum pattern matching fast on average. Algorithmica 77(1):16–39
Montanaro A (2020) Quantum speedup of branch-and-bound algorithms. Phys Rev Res 2(1):013056
Mozilla (2019) Common voice
Mozilla (2019) DeepSpeech
McClean JR, Romero J, Babbush R, Aspuru-Guzik Alán (2016) The theory of variational hybrid quantum-classical algorithms. New J Phys 18(2):23023
Nielsen MA, Chuang IL (2010) Quantum computation and quantum information. Cambridge University Press, Cambridge
Oudinet J, Denise A, Gaudel M-C (2013) A new dichotomic algorithm for the uniform random generation of words in regular languages. Theor Comput Sci 502:165–176
Piantadosi ST (2014) Zipf's word frequency law in natural language: A critical review and future directions. Psychon Bull Rev 21(5):1112–1130
Ponty Y (2012) Rule-weighted and terminal-weighted context-free grammars have identical expressivity. Research report
Pratap V, Xu Q, Kahn J, Avidov G, Likhomanenko T, Hannun A, Liptchinsky V, Synnaeve G, Collobert R (2020) Scaling up online speech recognition using ConvNets. facebook research
Reinharz V, Ponty Y, Waldispühl J (2013) A weighted sampling algorithm for the design of RNA sequences with targeted secondary structure and nucleotide distribution. Bioinformatics 29(13):i308–i315
Rabin MO, Scott D (1959) Finite automata and their decision problems. IBM J Res Dev 3(2):114–125
Stella M, Brede M (2016) Investigating the phonetic organisation of the English language via phonological networks, percolation and markov models. pp 219–229
Schmidhuber J (2014) Deep learning in neural networks: an overview. Neural Netw 61:85–117
Shor PW (1999) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev 41(2):303–332
Ilya S, Martens J, Hinton G (2011) 1017–1024. In: Proceedings of the 28th international conference on international conference on machine learning, ICML'11. Omnipress
Oliveira DS, Ramos R (2007) Quantum bit string comparator: circuits and applications. Quantum Comput Comput 7:01
Steinbiss V, Tran B-H, Ney H (1994) Improvements in beam search. In: Third international conference on spoken language processing
Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds) Advances in neural information processing systems, vol 27. Curran Associates, Inc., pp 3104–3112
Tang E (2019) A quantum-inspired classical algorithm for recommendation systems. In: Proceedings of the 51st Annual ACM SIGACT symposium on theory of computing - stoc 2019.ACM Press, New York, pp 217–228
Thompson K (1968) Programming techniques: regular expression search algorithm. Commun ACM 11(6):419–422

Ullman AVA, Lam MS, Sethi R, Jeffrey D (1997) Construction of an NFA from a regular expression. PWS Pub. Co, Boston

Vijayakumar AK, Cogswell M, Selvaraju RR, Sun Q, Lee S, Crandall D, Batra D (2016) Diverse beam search: decoding diverse solutions from neural sequence models. pp 1–16

Apeldoorn JV, Gilyén A, Gribling S, de Wolf R, Gilyen A, Gribling S, de Wolf R (2017) Quantum SDP-solvers: better upper and lower bounds. In: Annual symposium on foundations of computer science - Proceedings, 2017-Octob(617), pp 1–74

Vilares D, Gómez-Rodriguez C (2018) Transition-based parsing with lighter feed-forward networks. UDW@EMNLP

Wiebe N, Bocharov A, Smolensky P, Troyer M, Svore KM (2019) Quantum language processing

Wang D, Higgott O, Brierley S (2019) Accelerated variational quantum eigensolver. Phys Rev Lett 122(14):140504

Wiseman S, Rush AM (2016) Sequence-to-sequence learning as beam-search optimization. In: Proceedings of the 2016 conference on empirical methods in natural language processing. Association for Computational Linguistics, Stroudsburg, pp 1296–1306

Wossnig L, Zhao Z, Prakash A (2018) Quantum linear system algorithm for dense matrices. Phys Rev Lett 050502:120

Yang Y, Huang L, Ma M (2018) Breaking the beam search curse: a study of (re-)scoring methods and stopping criteria for neural machine translation. In: Proceedings of the 2018 conference on empirical methods in natural language processing. Association for Computational Linguistics, Stroudsburg, pp 3054–3059

Younger DH (1967) Recognition and parsing of context-free languages in time n3. Inf Control 10(2):189–208

Zhang Y, Clark S (2008) A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In: Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics, pp 562–571

Zhang Y, features JoakimNivre. (2011) Transition-based dependency parsing with rich non-local. In: Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies: short papers, vol 2. Association for Computational Linguistics, pp 188–193

Zhu C, Qiu X, Huang X (2015) Transition-based dependency parsing with long distance collocations. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)