Washington University in St. Louis
Washington University Open Scholarship

Arts & Sciences Electronic Theses and Dissertations

Arts & Sciences

5-8-2024

Determination of output composition in reaction-advection-diffusion systems and improving language model performance with Re-Tuning

Eric Joseph Pasewark
Washington University in St. Louis

Follow this and additional works at: https://openscholarship.wustl.edu/art_sci_etds

Recommended Citation

Pasewark, Eric Joseph, "Determination of output composition in reaction-advection-diffusion systems and improving language model performance with Re-Tuning" (2024). *Arts & Sciences Electronic Theses and Dissertations*. 3052.
https://openscholarship.wustl.edu/art_sci_etds/3052

This Dissertation is brought to you for free and open access by the Arts & Sciences at Washington University Open Scholarship. It has been accepted for inclusion in Arts & Sciences Electronic Theses and Dissertations by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.

WASHINGTON UNIVERSITY IN ST. LOUIS

Department of Mathematics

Dissertation Examination Committee:

Renato Feres, Chair

John McCarthy

Donsub Rim

Ari Stern

Gregory Yablonsky

Determination of Output Composition in Reaction-Advection-Diffusion Systems and Improving Language Model Performance with Re-Tuning

by

Eric Pasewark

A dissertation presented to
Washington University in St. Louis
in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

May 2024

St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

Many thanks to my advisor, Renato Feres, who gave me the freedom to work on interesting problems and gave me lots of support during my PhD. My parents, for always being there for me. My partner, Isabella, for being a source of joy.

ABSTRACT OF THE DISSERTATION

Determination of Output Composition in Reaction-Advection-Diffusion Systems and Improving Language Model Performance with Re-Tuning

by

Pasewark, Eric

Doctor of Philosophy in Mathematics,

Washington University in St. Louis, 2024.

Professor Renato Feres, Chair

There are 2 main subjects studied in this thesis. The first is on modeling chemical reactions. We formulate the problem of determining how much product is formed from a reactor and model this problem using metric graphs. We develop an efficient method to explicitly solve the problem on metric graphs. We work through examples by hand and with code to solve the problem. The second subject is a novel method to improve language model performance on compositional tasks. Our method teaches a language model to break a given problem into different subproblems, create prompts for these, and then solve the subproblems in separate contexts. The model uses the solutions of the subproblems to return the solution to the original problem given to the model. This method improves performance on 3 common tasks in the length generalization literature.

1. Organization of the Thesis

This thesis presents work from 2 different papers: one studying a reaction-diffusion problem and the other studying language models. The work on the reaction-diffusion problem was completed with Renato Feres and Gregory Yablonsky. This makes up Part I, consisting of chapters 2-5. Chapter 2 presents an introduction to the reaction-diffusion problem and gives the basic setup for the problem. Chapter 3 presents the method to solve the problem. Chapter 4 shows examples of network reactors and their solutions. Finally, chapter 5 presents the mathematical theory behind the solution method and the reduction to network reactors. These chapters are largely taken from the paper "Determination of output composition in reaction-advection-diffusion systems on network reactors" [1] with Renato Feres and Gregory Yablonsky

The work on language models was completed with Kyle Montgomery, Kefei Duan, Dawn Song, and Chenguang Wang. This work is in a forthcoming paper that is currently under review. This makes up Part II, consisting of chapters 6-8. Chapter 6 starts with background to understand the length generalization setup and then explains our new method to improve performance in this setup. Chapter 7 presents experimental results of the method, showing superior performance on 3 common length generalization tasks. Chapter 8 discusses related work and possible extensions of Re-Tuning.

Additionally, the appendix 9, gives a list of symbols for the reaction-diffusion problem and additional information and results for the language model work.

The first part of the thesis is the product of many discussions and emails among myself, Renato Feres, and Gregory Yablonsky. Renato and I worked on the mathematical ideas behind this for more than 2 years and Gregory worked on the chemistry applications. Initially Renato and I wanted to look at optimizing catalyst positions based on the methods from Renato's previous papers. This involved multiple methods, including using PINNs [2] from machine learning (Donsub Rim also provided helpful discussions related to this). Along with this, we also pursued some other machine learning applications, including analyzing machine learning training dynamics from a stochastic process perspective, for example as in [3]. We also looked at applying similar techniques to analyze diffusion models [4]. This brought us to investigate using stochastic process methods to analyze the chemistry problem in Part I. However, we realized that using a PDE approach would be easier and ultimately went that direction. In the final product, I additionally contributed an existence and uniqueness proof using matrix methods, wrote the code to implement our method to solve for the output composition matrix, and used this for computational results in the thesis and for other analysis.

While Renato and I were looking at applications of math to machine learning and machine learning to math, language models started becoming much more powerful. I investigated some applications of language models to math and programming problems and realized that language models had difficulty solving many compositional problems. I thought that part of the problem may be that the language models have a difficult time ignoring information that is irrelevant to a current computational step. For example in adding 2 numbers, part of the process the language model may follow is to add individual digits of these numbers. However, the language model still attends to all of the other digits even if they are irrelevant to adding the current digits. This led me to develop a method to help the language model focus only on the current step of a

multi-step problem. I came up with the Re-Tuning method, implemented it in code, and ran initial experiments to analyze its performance on addition, parity, and dynamic programming. After it looked promising, I began working with my coauthors on this, Kyle Montgomery, Kefei Duan, Dawn Song, and Chenguang Wang. My coauthors ran additional experiments and we also collaborated to put all our ideas in writing. Additionally, Renato contributed many helpful discussions along the way.

2. Introduction to the Reaction-Diffusion Problem

2.1 Brief overview

We consider reaction-transport processes in open reactors in which systems of first order reactions involving a number of gas species and solid catalysts can occur at localized active regions. Reaction products flow out of the reactor into vacuum conditions and are collected at an exit boundary. The *output composition problem* (OCP) is to determine the composition (molar fractions) of the collected gas after the reactor is fully emptied. We provide a solution to this problem in the form of a boundary-value problem for a system of time-independent partial differential equations. We then consider *network-like reactors*, which can be approximated by a network consisting of a collection of nodes and 1-dimensional branches, with reactions taking place at nodes. For these, it is possible to solve the OCP in a simple and effective way, giving explicit formulas for the output composition as a function of the reaction coefficients and parameters associated with the geometric configuration of the system. Several examples are given in 4 to illustrate the method.

The main application of our work is to Temporal Analysis of Products (TAP) [5] experiments. In these experiments a reactor is filled with some inert material (something like sand) and contains one or more catalysts where a reaction can occur. Gas particles are injected at some

entrance to the reactor and products are retrieved at the exit. The gas moves by diffusion or advection, and may encounter the catalyst as they move. The gas retrieved at the exit may have some combination of the original reactants injected into the reactor and some of the products formed while the gas was in the reactor. A problem of interest is to determine the proportions of different chemical species retrieved at the exit. This is the output composition problem.

We follow inspiration from previous lines of work [5], [6], and model the 3D reactor as a metric graph. Roughly speaking, a metric graph is a regular mathematical graph with vertices and edges, along with the additional structure that each edge represents a continuous interval. With this model, we can explicitly solve the output composition problem. Additionally the solution is simple to implement and fast to run. For the experiments in Chapter 4, solving the output composition problem for each graph took less than a second.

2.2 Motivation

Reaction-transport problems, in particular reaction-diffusion problems, are among the most topical and widely studied in chemical engineering. They have been posed and investigated in classical works of chemical engineering since the very beginning of this discipline. See [7, 8]. In approaching such problems, different elements should be considered:

- The chemical reaction is complex, i.e., it consists of a set of reaction steps involving a reactive mixture.
- Chemical reactions are always accompanied by transport.
- Two types of transport are typically considered: “forced” propagation (advection) and/or “self-propagation” (diffusion).

- Reactions occur over the surface of catalytic units (pelets, particles, etc.), which may be either porous or non-porous.

- The distribution of catalytic units within the reactor space is non-uniform.

There is presently no general theory that addresses all these elements together, especially the last one, regarding the geometric configuration of the catalytic units distributed within the reactor space and separated by the non-active material. For a few references on this general topic we mention the classic works [7, 8] as well as [9]. In the present work we propose to address this need by modeling the geometric configuration in terms of a network structure, in the context of systems of linear reactions. The following sections of this introduction explain our general set-up, with detailed definitions given in subsequent sections.

In open reactors, one specific problem of basic interest is the determination of the composition of the reactive mixture after the reactor is fully evacuated, for a given mixture introduced initially. Such "injection-evacuation" operation is one of the basic procedures in chemical engineering, and it may take many forms. For example, a similar problem was studied within the Temporal Analysis of Products (TAP) approach [10–13]. Also the problem analysed in [14], concerning complex catalytic reactions accompanied by deactivation, is of this kind, where catalytic deactivation may be regarded as equivalent to reactor outflow in the present context. In such situations, a basic problem is to determine the composition of this output mixture.

In our network setting, this *output composition problem* (OCP) can be analysed in a computationally effective way. This analysis is the main concern of the thesis.

The common theoretical approach for obtaining output composition is to begin by solving the reaction-transport equations, from which one obtains the exit flow of substances from the reactor

and, through integration in time, the amounts of the reaction products. We show, however, that the output composition can be expressed directly as the solution to a boundary value problem for a time-independent system of partial differential equations, thus by-passing the technically more difficult analysis based on first solving the reaction-transport equations. In the setting of network reactors (to be introduced shortly) the boundary-value problem for the OCP reduces to a system of linear algebraic equations from which we can in many cases obtain explicit solutions in a straightforward manner.

It has long been noted that time-integral characteristics (or moments) of reactor outflow provide important information about the reaction-transport system, such as the determination of conversion from kinetic data. Early works in this direction are by Danckwerts and Zwietering [15–17], where the authors used non-reactive tracers for this purpose. In Danckwerts's approach ([15, 16]), a moment-based analysis of reactor output was used. By injecting a radioactive tracer at the reactor inlet and measuring its concentration at the outlet, Zwietering ([17]) showed that mixing patterns can be determined. In a similar vein is *Temporal Analysis of Products* (TAP) approach [14, 18–21]. In TAP studies, an insignificant amount of chemical reactants is injected at the reactor inlet and the resulting response at the outlet is subsequently analysed. Here again, a moment-based technique for the calculation of time-integral characteristics of reactor outflow is used. In many TAP systems, Knudsen diffusion is the only transport mechanism.

The novelty in our approach, as already noted, is that we seek an effective method for computing integral characteristics in reaction-transport systems directly, without the need to first solve the (linear) reaction-transport equations.

At the core of our analysis is a matrix $f(\mathbf{x}) = (f_{ij}(\mathbf{x}))$, which we call the *output composition matrix*, where the indices i, j label the substances involved in the reaction-transport process.

This matrix is defined as the molar fraction of substance labeled by j in the reactor output composition given that a unit amount of i is initially supplied to the system at position $\mathbf{x}$ in the reactor. On networks, substances are injected at nodes, often denote by n below. In that setting, we normally write $f(n)$. Most of the present part of the thesis is about the characterization and computation of the output composition matrix, and the mathematical justification of our method for computing it.

If the input mixture at node n has composition given by the vector of fractions $\alpha = (\alpha_1, \ldots, \alpha_N)$, where N is the number of substances and the sum of the α_i is 1, then the output composition is $\beta = (\beta_1, \ldots, \beta_N)$ such that

$$\beta_i = \sum_{j=1}^{N} \alpha_j f_{ji}(n).$$

Thus the output composition matrix summarizes all the information about the reaction-transport system that bears on the determination of the composition of output mixture, disregarding time-dependent characteristics of the output flow. A preliminary formulation of our main result is given in the next section.

Perhaps an analogy is useful in explaining the utility of the output composition matrix $f(n)$. It plays for linear reaction-transport systems a similar role to that of scattering operators in quantum theory: we may think of the different chemical species as different scattering channels; if we probe the system by injecting i at node n, the output in channel j is given by the matrix coefficient $f_{ij}(n)$.

The present work continues the line of investigation initiated in [6] and [5], where they authors considered the irreversible reaction $A \to B$ on networks and studied reaction yield as a function of the network configuration and reaction/diffusion coefficients. That study was

based on the Feynman-Kac formula and stochastic analysis. In addition to greatly extending this previous work by allowing much more general systems of reactions, we have replaced the stochastic analysis with a more concrete and elementary, and perhaps conceptually more transparent, approach entirely based on an analysis of the initial/boundary-value problem for the reaction-transport equations.

Our approach admits natural generalizations that we did not want to pursue here. For example, it is possible to refine the output composition matrix so that it gives amounts of reaction products that evacuated at different parts of the reactor exit boundary.

A computer program for the computation of output composition based on the main theoretical result of this part of the thesis is available at

`https://github.com/Pasewark/Reaction-Diffusion-output-composition/tree/main`.

2.3 General set-up and formulation of the output composition problem

The purpose of this work is to provide an effective answer to the following problem. Let us consider a system of reaction-advection-diffusion equations with linear reaction rate functions, taking place inside a reactor represented by $\mathcal{R}$, which may be imagined for the moment (precise definitions will be given shortly) as a region in 3-space partially enclosed by walls that are impermeable to the flux of gases—to be referred to as the *reflecting boundary* of $\mathcal{R}$—but still open at certain places to an exterior that is kept at vacuum conditions.

Thus the two-dimensional boundary of $\mathcal{R}$ is the union of two parts: the reflecting boundary and the *exit boundary*. The interior of $\mathcal{R}$ is permeable to gas transport and contains a distribution

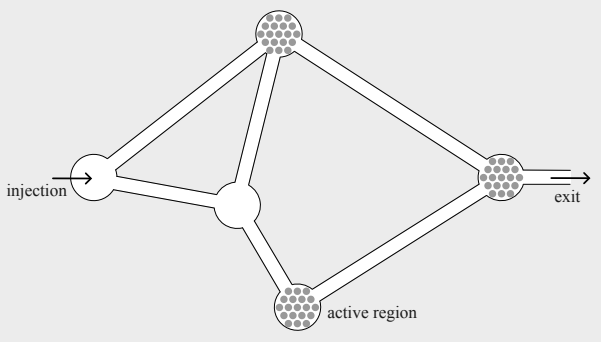


Figure 2.1.: A network-like reactor consists of pipes connected at junctures, which are places at which reactions may occur. The reactor interior is permeable to gas transport. Active regions contain catalyst particles that promote reactions involving the gas species.

of solid catalysts promoting first order reactions among N gaseous chemical species. We designate by $1, 2, \cdots, N$ the various species and by $\kappa_{ij}(\mathbf{x})$ the kinetic reaction coefficient for the linear reaction $i \to j$ at position $\mathbf{x}$. (Notice that our reactions are, formally, a system of isomerizations, although more general types can be accommodated by our analysis as will be indicated in Section 3.3.)

We have in mind situations in which the $\kappa_{ij}(\mathbf{x})$ are nonzero only at relatively small *active regions*. Diffusion coefficients and advection velocity fields are also specified for each gas species. Given this system configuration, we suppose that a mixture of these gas reactants is injected into $\mathcal{R}$ at known places and reaction products are collected at the exit boundary. The precise mode of injection does not need to be specified, although it is assumed that substance amounts are sufficiently small that the linear character of transport is attained early on in the process. After sufficient time, the reactor empties out and the full amount of reaction products is collected.

It will be shown that this *output composition problem*—abbreviated OCP—can be solved by a boundary-value problem for a system of time-independent partial differential equations.

We are particularly interested in reactors that have a network configuration, as suggested by Figure 2.1. That is, they consist of pipes linked to each other at *junctures* of relatively small volume. The whole ensemble will be called a *network-like reactor*. Reactions can only take place at junctures. Pipes are characterized by their cross-sectional areas, diffusivity and advection velocity, possibly dependent on species index i but constant along pipe cross-section; and the junctures are characterized by the functions $\kappa_{ij}(\mathbf{x})$, possibly equal to 0.

Given this information, and assuming that gas concentrations in the pipes are essentially constant along cross-sections, which is to be expected if pipes are long and narrow and diffusivity and advection velocities are also constant on cross-sections, the network-like reactor $\mathcal{R}$ becomes, effectively, 1-dimensional (see Figure 2.2). In this setting, we refer to pipes as *branches*, junctures as *nodes*, and $\mathcal{R}$ as a *network reactor* (dropping the suffix "like"). In this network setting, we show that the boundary-value problem referred to above can be solved with relative ease.

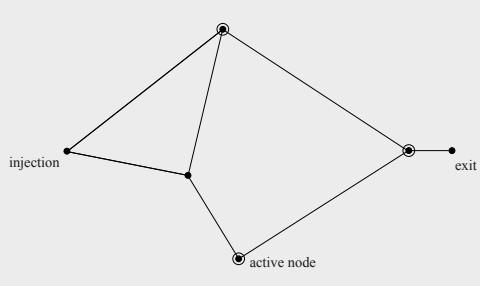


Figure 2.2.: Network reduction of the network-like reactor of Figure 2.2.

By an effective answer to this output composition problem we have in mind formulas for the amount of each gas species in the reactor output showing the explicit dependence on reaction coefficients at the various active nodes, with coefficients that are known functions of the geometric/topological parameters of the network such as lengths of branches and degrees of nodes, initial data, and the coefficients of diffusivity and advection

velocities. (The latter transport parameters will be assumed constant along pipes in our examples.) As will be seen, the output composition problem for network reactors can be conveniently expressed in terms of a matrix $f(n) = (f_{ij}(n))$ whose entry $f_{ij}(n)$ is the fraction of species j in the output given that a unit pulse of i is injected into the reactor at node n. When the transport parameters are constant along branches, this quantity will turn out to be a rational function of the reaction rate constants with coefficients that are polynomial functions of the *velocity-adjusted lengths* of the network branches, a measure of the effective length of pipes to be defined shortly.

2.4 Organization of the reaction-diffusion chapters

Chapters 3-5 are organized as follows. In Chapter 3, we describe the system of partial differential equations and initial/boundary-value conditions that serve as the mathematical model for the reaction-transport process and show the boundary-value problem that solves the OCP for general reactor domains. (3.1.) We then introduce definitions and notation needed to describe the network version of the OCP (3.2). In 3.3 we make a few remarks about the nature of the reactions this study is restricted to and in 3.4 we describe the relationship between reaction coefficients on the network-like reactor and on its network approximation. The matrix $f(n)$ encoding the solution to the OCP is explained in greater detail in 3.6 and the method for obtaining it as solution of a system of linear algebraic equations is shown in Chapter 3. It turns out that the presence of advection velocities enters into $f(n)$ in a very simple way through the introduction of *velocity-adjusted lengths* defined and explained in 3.7. Chapter 4 provides several examples to illustrate the procedure for solving the OCP and makes a few observations about the nature

of solutions. In Chapter 5 we present the mathematical proof of the main result for general reactor domains and in 5.4 we show how those results are formulated for network reactors. In the appendix, 9.1, we provide a glossary of the most frequently used symbols.

3. Definitions and method of solution

3.1 The boundary-value problem solving the OCP

For the moment, let us assume that the reactor $\mathcal{R}$ is a general domain in coordinate 3-space with nice (differentiable) reflecting and exit boundaries. The mathematical model for the reaction-transport system is as follows. Let $D_i(\mathbf{x})$ and $\mathbf{v}_i(\mathbf{x})$ represent the diffusivity and advection velocities, which we allow to depend on the gas species. These species are labeled by $i = 1, \ldots, N$. The concentrations $c_i(\mathbf{x}, t)$ satisfy the system of equations (see, for example, [7])

$$\frac{\partial c_i}{\partial t} + \nabla \cdot \mathbf{j}_i = \sum_j c_j \kappa_{ji}, \tag{3.1}$$

where $\mathbf{j}_i = c_i \mathbf{v}_i - D_i \nabla c_i$ is the flux vector field of species i, and boundary conditions:

- $c_i(\mathbf{x}, t) = 0$ for $\mathbf{x}$ on the exit boundary of $\mathcal{R}$;
- the normal component of the flux, $\mathbf{n}(\mathbf{x}) \cdot \mathbf{j}_i(\mathbf{x}, t)$, equals zero on the reflecting boundary.

Here $\mathbf{n}(\mathbf{x})$ is the unit normal vector pointing outward at a boundary point $\mathbf{x}$ of $\mathcal{R}$. One further specifies initial concentrations at time $t = 0$.

In the long run, all reaction products (including gas that didn't undergo reaction) leave $\mathcal{R}$ through the exit boundary. The amount of each species in the reactor output is the quantity of

primary interest in our analysis. Since the initial-boundary-value problem is linear, in order to predict the output composition it is sufficient to determine the quantities $f_{ij}(\mathbf{x})$ defined as:

$$f_{ij}(\mathbf{x}) = \text{ fraction of } j \text{ in the output given that a unit pulse of } i \text{ is initially injected at } \mathbf{x}. \quad (3.2)$$

We call $f(\mathbf{x}) := (f_{ij}(\mathbf{x}))$ the *output composition matrix* for each injection point $\mathbf{x}$. Our analysis begins with the observation that this matrix-valued function on $\mathcal{R}$ satisfies the system of partial differential equations

$$\nabla \cdot (D_i \nabla f_{ij}) + \mathbf{v}_i \cdot \nabla f_{ij} + \sum_k \kappa_{ik} f_{kj} = 0, \quad j = 1, \ldots, N, \tag{3.3}$$

and boundary conditions

$$\begin{aligned} \mathbf{n} \cdot \nabla f_{ij} &= 0 \quad \text{on the reflecting boundary of } \mathcal{R} \\ f_{ij} &= \delta_{ij} \quad \text{on the exit boundary of } \mathcal{R}. \end{aligned} \tag{3.4}$$

This result will be proved in Chapter 5. Under a few simplifying assumptions that are natural to network-like reactors, it is possible to solve the boundary-value problem explicitly in many cases. In the network approximation, the reactor domain $\mathcal{R}$ becomes a union of (possibly curved) line segments that we call *branches*, which are connected to each other at points called *nodes*. We further assume that the transport quantities (diffusivities and advection velocities) are constant along branches and chemically active regions are reduced to nodes. The reduction of the OCP and its solution to network reactors will be detailed in Section 5.4.

We begin this analysis in the next section by introducing some notation and terminology for reaction-transport processes on networks.

3.2 Some network definitions and notation

As already indicated, the network-like reactor, consisting of thin pipes connected to each other at junctures, with chemically active regions restricted to junctures, will be replaced with an actual network (or graph) consisting of 1-dimensional branches (edges) and point nodes (vertices). All the geometric, transport and reaction parameters relevant to the output composition problem will become parameters assigned to these branches and nodes. The following description refers to this network reduction, which we will call the *network reactor*, or simply the *network*, and continue to denote by $\mathcal{R}$. Figure 3.1 will be used to illustrate the main definitions introduced in this section.

Proceeding more formally, we define a network reactor $\mathcal{R}$ as a union of finitely many lines in coordinate 3-space with finite length, called the reactor's *branches* and indicated by the labels $b_0, b_1, \ldots$. These branches are joined at points which we call the reactor's *nodes*, indicated by $n_0, n_1, \ldots$. (Indexing branches and nodes starting from 0 is, of course, an arbitrary matter. Occasionally, we begin from 1.) Each branch b can be oriented in two possible ways, indicated by $\mathbf{b}$ and $\overline{\mathbf{b}}$. It is useful to make from the beginning an arbitrary choice of orientation for each b (indicated by arrows in the network diagrams) so that branch velocities can be expressed by a (positive or negative) number attached to b. We may occasionally use the opposite branch orientation than the one indicated by the arrow. This may happen, for example, when writing node conditions for our differential equations, where it is convenient to orient branches attached to a given node in the direction pointing away from the node. In such situations, the sign of the advection velocity is flipped. An oriented branch $\mathbf{b}$ may also be indicated by the pair of its initial and terminal nodes, (n, n'). This can only be done when there is at most one branch

between any pair of nodes. By adding additional inert nodes, this assumption can be made without any loss of generality and without changing the properties of the system. We then have $\overline{(n, n')} = (n', n)$.

To summarize, branches and nodes of the network reactor $\mathcal{R}$ are assigned the following set of parameters:

- To each branch b is associated its length $\ell(b)$, diffusivity coefficients $D_i(b)$ (also indicated by D_i^b at some places in the analysis) where $i = 1, \cdots, N$ labels the gas species, and advection velocities $\nu_i(\mathbf{b})$. Velocities can be positive, negative or zero and we have $\nu_i(\overline{\mathbf{b}}) = -\nu_i(\mathbf{b})$. When branch orientations are explicitly indicated on network diagrams by arrows (as we do in the examples), we may simply write $\nu_i(b)$ or ν_i^b.

- To each node n we associate a matrix $K(n) = (K_{ij}(n))$ where $K_{ij}(n)$ is the constant of the $i \to j$ reaction at node n. It is mathematically convenient to define $K_{ii}(n)$ to be equal to the negative of the sum of the $K_{ij}(n)$ for all j not equal to i. Defined this way, the sum of the entries of each row of $K(n)$ is 0. Notice that we are using different symbols for the reaction coefficient $\kappa_{ij}(\mathbf{x})$ for a general (network-like) reactor and $K_{ij}(n)$ for the corresponding constant on the network reactor approximation. The latter is obtained from the former by a scaling limit and so they are, as we will see shortly (section 3.4), distinct quantities. It is the $K_{ij}(n)$ (and not the $\kappa_{ij}(\mathbf{x})$) that will appear in our explicit formulas for output composition on network reactors.

- To each node n and each branch b attached to n we define $p(n, b)$ as the ratio of the cross-sectional area of the pipe (of the original network-like domain, represented by b in the network reduction) over the sum of the cross-sectional areas of all the branches

attached to n. So defined, the sum of the $p(n,b)$ for a given n is 1. In the examples, we always take $p(n,b) = 1/\deg(n)$, where the degree of a node, $\deg(n)$, is the number of branches connected to it. This amounts to the assumption that all pipes have the same cross-sectional area.

- We may identify an oriented branch $\mathbf{b}$ with the closed interval $[0, \ell(b)]$. More precisely, we associate to $\mathbf{b} = (n, n')$ a (twice continuously differentiable) parametrization of b by arc-length, $\varphi_{\mathbf{b}}(x)$, where $0 \le x \le \ell(b)$, so that $\varphi_{\mathbf{b}}(0) = n$ and $\varphi_{\mathbf{b}}(\ell(b)) = n'$. We are thus indicating points along b by their distance (length) from the initial node. If f is any function on $\mathcal{R}$, its restriction to a branch b becomes a function of x: $f^{\mathbf{b}}(x) := f(\varphi_{\mathbf{b}}(x))$. The reason for writing $\mathbf{b}$ as a superscript is that such functions will typically be further indexed by the label of the chemical species. For example, the concentration of i at the point $\varphi_{\mathbf{b}}(x)$ may be written $c_i^{\mathbf{b}}(x) := c_i(\varphi_{\mathbf{b}}(x))$. Notice that the derivative $\varphi'_{\mathbf{b}}(x)$ is a unit length vector. If $\mathbf{v}_i$ is the advection velocity of i, then $\mathbf{v}_i(\varphi_{\mathbf{b}}(x)) = \nu_i^{\mathbf{b}}(x)\varphi'_{\mathbf{b}}(x)$. (We are not yet assuming that advection velocities and diffusivities are constant along branches.)

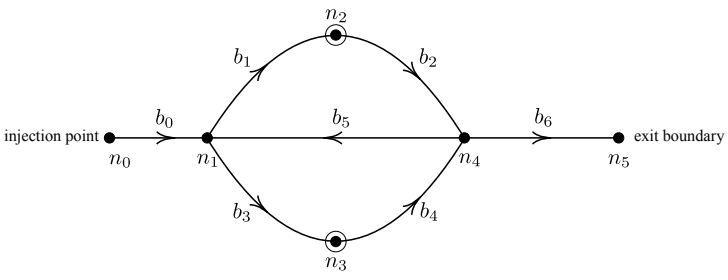


Figure 3.1.: The main elements of a network reactor. Active nodes are indicated by a circle around a solid dot.

A node will be called (chemically) *active* if some reaction coefficient at it is positive. An *inactive* node is one at which all reaction coefficients equal 0. A subset of inactive nodes will constitute the *exit boundary*. In the example of Figure 3.1, this is the single-point set con-

sisting of n_5. The exit boundary is

where products can leave the reactor and where concentrations are set equal to zero (vacuum conditions). We call *internal* nodes those nodes that do not lie in the exit boundary. A collection of internal nodes, active or inactive, may be chosen for the place at which an initial pulse of reactants is injected into the reactor.

In practice, it is useful to allow the exit boundary to consist of multiple nodes but, mathematically, no generality is lost if we assume that all these exit nodes merge into a single one. It is also convenient, as already noted, to suppose that, between each pair of nodes, there can be at most one connecting branch, so that an oriented branch $\mathbf{b}$ can also be indicated by the pair (n, n') of initial and terminal nodes. When describing vectors (advection velocities and gradients of functions) both notations, say $\nu_i(\mathbf{b})$ or $\nu_i(n, n')$, may be used depending on convenience and context.

3.3 General reactions with linear rate functions

We have assumed that reactions are linear, of the type $i \to j$, but our analysis can accommodate more general types so long as rate functions are linear. This is explained below.

For the purposes of this section, we regard chemically active junctures in the network-like reactor as CSTRs with in and out fluxes where junctures connect to pipes. Let us suppose that the reaction mechanism on a given juncture consists of a set of S reactions—generally an even number since we count separately a reaction and its reverse—involving N molecular species.

We denote these species here by $A_1, \ldots, A_N$. (Elsewhere in the thesis, they are indicated simply by i rather than A_i.) Let each reaction, labeled by the index $s = 1, \ldots, S$ and given by

$$a_{s1}A_1 + \cdots + a_{sN}A_N \to b_{s1}A_1 + \cdots + b_{sN}A_N,$$

have reaction rate functions $w_s(c)$ where $c = (c_1, \ldots, c_N)$. The stoichiometric coefficients, a_{si}, b_{si}, are non-negative integers and $a_{si}b_{si} = 0$ so that no species appear on both sides of the reaction equation. The rate of change of the concentration of A_i is

$$\frac{dc_j}{dt} = \sum_{s=1}^{S}(b_{sj} - a_{sj})w_s(c) + \text{net flux of } A_j \text{ in and out of juncture.}$$

We assume that the rate function $w_s(c)$ is a linear function of the concentration of one of the input species (that is to say, one of the A_i for which a_{si} is not zero). This approximation is often made in two cases: (a) the actual reaction mechanism involves a linear step whose rate coefficient is significantly smaller than those of the other reaction steps; in such a case, the overall rate can be dominated by that of the slower linear reaction. (b) The reaction is bimolecular and involves components A_1 and A_2, A_2 being much more abundant than A_1. In this case, the reaction rate can often be approximated by a linear function of the concentration of A_1. We refer to [9] for more details on such issues in chemical kinetics. Generally, such linearization is an important problem in chemical engineering, but it is an issue that lies outside the scope of the thesis.

Thus we assume that $w_s(c) = w_{i_s}c_{i_s}$ for some i_s. By introducing

$$\kappa_{ij} := \sum_{s=1}^{S} w_{is}(b_{sj} - a_{sj})\delta_{ii_s}$$

we reduce these differential equations to our preferred form:

$$\frac{dc_j}{dt} = \sum_{i=1}^{N} c_i\kappa_{ij} + \text{net flux of } A_j.$$

In our analysis of the OCP, we rely on general facts about solutions of systems of parabolic partial differential equations that require further assumptions on the entries of the matrix κ, the most important being that the off-diagonal entries should be non-negative. (See, for example, [22] 3.4.1.) Reaction equations for which these assumptions most directly apply are of the type $A_i \rightleftharpoons A_j$. However, it is possible to accommodate more general types whose rates are still linear. Consider, for example, the situation in which species A_1, A_2, A_3 are involved in the irreversible reaction

$$2A_3 \to A_1 + A_2$$

with rate $w(c) = wc_3$. In this case, $i_s = 3$. We then have $\kappa_{ij} = 0$ if $i \neq 3$ and the third row of κ is $w(1, 1, -2)$. Observe that this system is equivalent to two irreversible isomeric reactions: $A_3 \to A_1$, $A_3 \to A_2$. For another example, let us consider the system of three reaction pairs:

$$2\,\mathrm{A}_1 \underset{\mathrm{w}_-^{(1)}}{\overset{\mathrm{w}_+^{(1)}}{\rightleftharpoons}} 2\mathrm{A}_3, \quad 2\,\mathrm{A}_2 \underset{\mathrm{w}_-^{(2)}}{\overset{\mathrm{w}_+^{(2)}}{\rightleftharpoons}} 2\mathrm{A}_3, \quad \mathrm{A}_1 \underset{\mathrm{w}_-^{(3)}}{\overset{\mathrm{w}_+^{(3)}}{\rightleftharpoons}} \mathrm{A}_2.$$

Here the rate functions are $w_+^{(1)}c_1$ and $w_-^{(1)}c_3$ for the first pair, $w_+^{(2)}c_2$ and $w_-^{(2)}c_3$ for the second, and $w_+^{(3)}c_1$ and $w_-^{(3)}c_2$ for the third. In this case,

$$\kappa = \begin{pmatrix} -\left(2w_+^{(1)} + w_+^{(3)}\right) & w_+^{(3)} & 2w_+^{(1)} \\ w_-^{(3)} & -\left(w_-^{(3)} + 2w_+^{(2)}\right) & 2w_+^{(2)} \\ 2w_-^{(1)} & 2w_-^{(2)} & -2\left(w_-^{(1)} + w_-^{(2)}\right) \end{pmatrix}.$$

Notice that the off-diagonal elements of this matrix are non-negative and the sum of the elements of each row is 0. These are the two properties of κ that we assume for the main results of the thesis.

It is not clear to us how general the matrix κ can be for our analysis to still be valid. This analysis can be significantly extended to treat more general (multimolecular) reactions than

considered so far, even if many of them may not be meaningful from a chemical engineering perspective (although they could be relevant in other areas of science where kinetic models are used). It is an interesting problem to determine how general a system of reactions with linear rates we can accept, but this article is not the place to deal with such mathematical issues. In all the examples discussed below, we have limited ourselves to reactions of type $A_i \rightleftharpoons A_j$.

3.4 The relationship between $\kappa_{ij}(\mathbf{x})$ and $K_{ij}(n)$

Next, we need to understand the relationship between the reaction coefficients $\kappa_{ij}(\mathbf{x})$ on the network-like reactor in dimension 3 and the corresponding constants $K_{ij}(n)$ defined at a node n on the network reduction. Recall that on the network-like reactors, active regions are localized but still 3-dimensional. In the passage to network reactors these regions shrink down to points. This entails a change of physical units, so that $K_{ij}(n)$ has units of distance over time, not the reciprocal of time. The purpose of the present section is to provide some clarification on this issue.

When we get to see explicit formulas for output composition later in this part of the thesis, we will often encounter dimensionless expressions of the form $\ell K/D$, where ℓ is a length, D a constant of diffusivity, and K a reaction coefficient associated with an active node. This may seem at first to conflict with standard quantities in chemical engineering. Notice that κ, with the physical unit reciprocal of time, is a sort of "density of chemical activity." If this activity is strongly localized along a small interval of length ϵ on a network branch, one needs to integrate over this length to obtain activity on that small interval. Then κ should scale as $\kappa_\epsilon \approx K/\epsilon$ and $\ell K/D \approx \ell\epsilon\kappa_\epsilon/D$.

More precisely, let us look at what happens in a neighborhood of an active juncture. The network approximation of the network-like reactor made of pipes and junctures will depend on a small length parameter ϵ. This is the radius of a ball centered at any node n that delimits the chemically active region at that juncture, if the juncture is active, and the part of the network near n having a relatively complicated geometry, as opposed to the simple tube-like shapes on the complement of the union of junctures. See Figure 3.2. Outside of such balls, pipes have uniform cross-section and no reactions take place. This same ϵ is assumed to hold for all the nodes and the smaller the value of ϵ the better is the network approximation.

Since, in the network model, active regions shrink to points, the reaction coefficients $\kappa_{ij}(\mathbf{x})$ at any position $\mathbf{x}$ should be scaled up with the reciprocal of ϵ. This is because the probability that a molecule will react is proportional to the total amount of time it spends in the active regions; as these regions shrink to a point, the rate constants should scale up accordingly. We indicate this by writing $\kappa_{ij}(\epsilon, \mathbf{x})$.

Let $V_\epsilon(n)$ and $A_\epsilon(n)$ be, respectively, the volume of the juncture $\mathcal{R}_\epsilon(n)$ and the area of the part of the boundary of this juncture complementary to the reflecting boundary—a union of pipe cross-sectional discs. In taking the limit as ϵ approaches 0, we suppose that the dimensionless quantity $\epsilon A_\epsilon(n)/V_\epsilon(n)$ converges to the positive quantity $\chi(n)$, a geometric characteristic of the juncture that survives the limit process, and

$$\frac{1}{A_\epsilon(n)} \int_{\mathcal{R}_\epsilon(n)} \kappa_{ij}(\epsilon, \mathbf{x})\, dV(\mathbf{x}) \to K_{ij}(n).$$

In the above volume integral, keep in mind that $\kappa_{ij}(\epsilon, \mathbf{x})$ is of the order $1/\epsilon$ and has physical dimension $1/$time. Thus $K_{ij}(n)$ has physical dimension length/time.

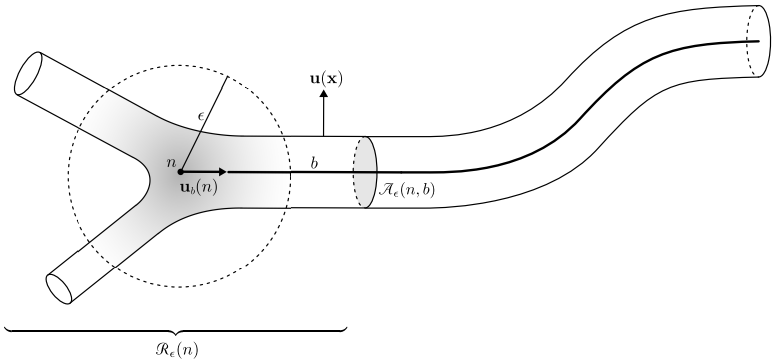


Figure 3.2.: A juncture $\mathcal{R}_\epsilon(n)$ in the network-like reactor corresponding to a node n with an attached pipe. The gradient shading represents catalyst distribution and ϵ is a radius delimiting the chemically active region. The unit vector $\mathbf{u}(\mathbf{x})$ is orthogonal to the *reflecting boundary* of $\mathcal{R}$ whereas $\mathbf{u}_b(n)$ is the unit vector orthogonal to the cross-section of the pipe b pointing away from n. $\mathcal{A}_\epsilon(n,b)$ is the cross-sectional disc where the pipe attaches to the juncture. Other features are explained in the text.

Also note the limit $A_\epsilon(n,b)/A_\epsilon(n) \to p(n,b)$, where $A_\epsilon(n,b)$—the area of $\mathcal{A}_\epsilon(n,b)$ (see Figure 3.2)—is the cross-sectional area of the pipe corresponding to the branch b attached to n.

These considerations will become important in Section 5.4.1, where we obtain node conditions for the composition output boundary-value problem.

3.5 The boundary-value problem for output composition

Our main result for network reactors, described in the next section, is a consequence of the already mentioned (see section 3.1) observation that the output composition matrix $f(\mathbf{x})$, for a not necessarily network-like reactor domain $\mathcal{R}$, is the solution to the following boundary-value problem:

$$(\mathcal{A}f)_{ij} := \nabla \cdot (D_i \nabla f_{ij}) + \mathbf{v}_i \cdot \nabla f_{ij} + \sum_k \kappa_{ik} f_{kj} = 0, \quad i,j = 1,\dots,N, \tag{3.5}$$

and boundary conditions

$$\begin{gathered} \mathbf{n} \cdot \nabla f_{ij} = 0 \quad \text{on the reflecting boundary of } \mathcal{R} \\ f_{ij} = \delta_{ij} \quad \text{on the exit boundary of } \mathcal{R}. \end{gathered} \tag{3.6}$$

To begin to develop an understanding of this boundary-value problem, let us consider two extreme cases: (a) the reaction coefficients are negligible compared to the transport coefficients; (b) the network-like reactor consists of one small active region directly open to the outside. In this case, there are only to positions to consider: inside and outside the reactor and the only relevant transport characteristic is the rate of evaculation of each substance.

In case (a), neglecting reactions ($\kappa_{ij}(\mathbf{x}) = 0$), the solution to the boundary-value problem is the constant matrix with elements $f_{ij}(\mathbf{x}) = \delta_{ij}$. The obvious interpretation is that the composition of the output mixture equals the composition of the initially injected mixture.

Let us now turn to special case (b). We don't have, in this case, the transport terms of the general reaction-transport equation, so it is necessary to introduce a rate of evacuation. We model this situation by imagining that space consists of two points, one representing the active region and the other the outside of the reactor (the surrounding of the active region). We then introduce irreversible reactions of type $A_i \mapsto X_i$, with a rate μ_i, where A_i represents a substance inside the reactor and X_i is the same substance outside, where $i = 1, \ldots, N$. Thus this "reaction-evacuation" process is mathematically equivalent to a reactions-only process inside a closed reactor (which we may imagine as a batch reactor) observed from time 0 until chemical equilibrium is reached and only the substances X_i remain. The matrix f for the reactions-only process has the form

$$f = \begin{pmatrix} f_{AA} & f_{AX} \\ f_{XA} & f_{XX} \end{pmatrix} = \begin{pmatrix} O & f_{AX} \\ O & I \end{pmatrix},$$

where the blocks have size $N \times N$, O is the zero matrix, and I is the identity. The notation for the block indices should be interpreted as follows: the (i, j)-element of f_{AX} gives the fraction of X_j at equilibrium when a unit amount of A_i is introduced initially. With similar interpretations, it is clear that $f_{AA} = f_{XA} = O$ and $f_{XX} = I$. The reaction coefficients can be similarly written in

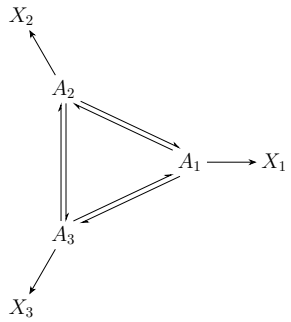


Figure 3.3.: A system of reactions with three terminal species (absorbing states).

block matrix form:

$$\kappa = \begin{pmatrix} \kappa_{AA} & \kappa_{AX} \\ \kappa_{XA} & \kappa_{XX} \end{pmatrix} = \begin{pmatrix} \kappa_{AA} & \kappa_{AX} \\ O & O \end{pmatrix},$$

where κ_{AX} is the diagonal matrix $\mathrm{diag}(\mu_1, \ldots, \mu_N)$. The concentrations c_i evolve in time according to the system of equations

$$\frac{dc_i}{dt} = \sum_j c_j \kappa_{ji}.$$

We wish to show that our boundary-value problem, in this special case, is indeed solved by the output composition matrix f. First observe that, in the absence of diffusion and advection, $\mathcal{A}f = \kappa f$, so Equation (3.5) reduces to $\kappa f = 0$, and the boundary condition (3.6) reduces to $f_{XX} = I$. Written in terms of the matrix blocks, these two equations amount to

$$\kappa_{AA} f_{AX} + \kappa_{AX} = 0. \tag{3.7}$$

In order to see that Equation (3.7) is, indeed, satisfied by f, we now introduce a key ingredient of the general proof that, in this special case (b), reduces to

$\rho_{ij}(t|s) =$ concentration of i at time t given that unit amount of j is introduced at time $s < t$.

Naturally,

$$\rho'_{ij}(t|0) = \sum_k \rho_{kj}(t|0)\kappa_{ki}, \quad \rho_{ij}(0|0) = \delta_{ij},$$

where $\rho'_{ij}(t|0)$ is derivative in t. We then have the following fundamental equation, which we accept here on heuristic grounds:

$$f_{ij} = \sum_k \rho_{ki}(t|0) f_{kj}. \tag{3.8}$$

In words, if a unit amount of i is introduced at time 0, then the final amount of j equals the final amount of j given that a unit amount of k is introduced at time t weighted by the

concentration of k at time t. Said yet differently, the output amount of j is the same whether we start with the composition consisting only of i at time 0, or with the mixture defined by $(\rho_{1i}(t|0), \ldots, \rho_{Ni}(t|0))$ at time t.

Differentiating in t at time 0 and using the equation and initial condition for $\rho_{ki}(t|0)$ yields

$$0 = \sum_k \kappa_{ki} f_{kj}.$$

In this system of equations, i represents one of the A-species and j one of the X-species. Breaking the sum into these two types of indices, we obtain the system of equations (3.7), which is what we wished to demonstrate.

The above remark brings into consideration the matrix-valued function ρ. Back to general reaction-advection-diffusion systems, a similar function plays a central role in the proof given in Chapter 5. The following remarks highlight the role of this key ingredient.

First notice that the total amount of substance i produced in the long run by the reaction-transport process in the open reactor $\mathcal{R}$ is (here dx is ordinary volume element):

$$\text{Amount of } i \text{ in gas output } = \text{Initial amount of } i \; + \underbrace{\sum_j \int_0^\infty \int_{\mathcal{R}} c_j(\mathbf{x}, t)\kappa_{ji}(\mathbf{x})\, dx\, dt}_{\text{net amount of } i \text{ produced by reactions}}.$$

We wish to view this quantity as a function of the initial mixture pulsed into the reactor at time 0. Let us introduce the *fundamental solution* to the reaction-transport equations:

$$\rho_{ij}(\mathbf{x}, t|\mathbf{y}, s) = \text{ concentration of } i \text{ at } (\mathbf{x}, t) \text{ given that a unit pulse of } j \text{ is injected at } (\mathbf{y}, s), s < t.$$

Then, using the previous balance equation, we obtain that the total amount of j in the gas output given that a unit of i is introduced at the beginning of the process is given by

$$f_{ij}(\mathbf{y}) = \delta_{ij} + \sum_k \int_0^\infty \left[\int_{\mathcal{R}} \rho_{ki}(\mathbf{x}, t|\mathbf{y}, 0)\kappa_{kj}(\mathbf{x})\, dx \right] dt.$$

In matrix form,

$$f(\mathbf{y}) = I + \int_0^\infty \int_{\mathcal{R}} \kappa^\top(\mathbf{x}) \rho(\mathbf{x}, \mathbf{t}|\mathbf{y}, 0)\, dx\, dt,$$

where $\kappa^\top$ indicates matrix transpose. One is led to ask for a boundary-value problem characterizing the time-independent function $f(\mathbf{y})$. We know that, as a function of $(\mathbf{x}, t)$ with $(\mathbf{y}, s)$ fixed, the quantity $\rho(\mathbf{x}, \mathbf{t}|\mathbf{y}, 0)$ satisfies the reaction-transport equation with initial pulse condition

$$\rho_{ij}(\mathbf{x}, t|\mathbf{y}, s) \xrightarrow{t \downarrow s} \delta(\mathbf{x} - \mathbf{y})\delta_{ij}.$$

The key fact we need, well-known in the theory of parabolic differential equations (see, for example, [23]), is that $\rho(\mathbf{x}, \mathbf{t}|\mathbf{y}, s)$, as a function of $(\mathbf{y}, s)$ for $(\mathbf{x}, t)$ fixed, satisfies a similar initial-boundary-value problem. Specifically, writing $\rho^*(\mathbf{y}, s|\mathbf{x}, t) = \rho(\mathbf{x}, t|\mathbf{y}, s)^\top$,

$$-\frac{\partial \rho^*}{\partial s} = \mathcal{A}\rho^*.$$

Here, the derivatives involved in $\mathcal{A}$ are relative to $\mathbf{y}$. (See Section 5.2 for details.) This is the place where the operator $\mathcal{A}$ finally enters the picture. The rest of the verification of the main claim of this section now follows from the relatively straightforward mathematical manipulations described in greater detail in Chapter 5. Details apart, we believe the conceptual core of this story lies in the fundamental identity

$$f_{ij}(x) = \sum_k \int_{\mathcal{R}} \rho_{ki}(y, t|x, 0) f_{kj}(y)\, dy, \tag{3.9}$$

generalizing Equation (3.8), which is very natural given the interpretation of the matrix $\rho(y, t|x, 0)$.

3.6 The output composition matrix for network systems

By the *output composition matrix* for network reactors, $f(n) = (f_{ij}(n))$, we mean a matrix-valued function of the node n having the following interpretation: $f_{ij}(n)$ is the fraction of gas species j in the reactor's output given that a unit pulse of species i is initially injected at node n. By definition, if n is an exit node, then $f(n) = I$ is the identity matrix; that is, $f_{ij}(n) = \delta_{ij}$, the Kronecker delta, which is 0 if $i \neq j$ and 1 otherwise.

Due to the linearity of the system of equations giving $f(n)$, if a mixture of gases is injected at n having composition vector $\alpha = (\alpha_1, \ldots, \alpha_N)$ where α_i is the molar fraction of i, then $\beta = \alpha f(n)$ is the vector of molar fractions in the reactor output. The fraction of j in the output composition is then

$$\beta_j = \sum_{i=1}^{N} \alpha_i f_{ij}(n).$$

More generally, if the fraction α_i of i is injected at node n_i, then $\beta_j = \sum_{i=1}^{N} \alpha_i f_{ij}(n_i)$.

The determination of $f(n)$ is our central problem. It will be shown in Chapter 5, for general reactor domains in 3-space, that this matrix-valued function satisfies a boundary value problem for a time-independent system of elliptic differential operators which, when reduced to network domains (in Section 5.4), amounts to Equations (3.10), (3.11) and (3.12) given below.

Summarizing the main result, the matrix $f(n)$, for each internal node n of the network reactor, is obtained as the solution to the following time-independent boundary-value problem (the network counterpart of Equations (3.3) and boundary conditions (3.4)). On each branch b:

$$D_i(b) f_{ij}''(x) + \nu_i(b) f_{ij}'(x) = 0. \tag{3.10}$$

Here $f'(x)$ indicates derivative with respect to the arc-length parameter x along b. On internal nodes,

$$\sum_{b\sim n} p(n,b)D_i(b)f'_{ij}(n,b) + \sum_k K_{ik}(n)f_{kj}(n) = 0, \tag{3.11}$$

where $b \sim n$ indicates that the sum is over those branches that are attached to node n, and $f'_{ij}(n,b)$ denotes the derivative at 0 of the restriction of f_{ij} to branch b in the arc-length parameter x of b oriented away from n (that is, so that $x = 0$ corresponds to n). Finally, on exit nodes n_{exit},

$$f_{ij}(n_{\text{exit}}) = \delta_{ij}. \tag{3.12}$$

Equations (3.10) together with boundary conditions (3.11) and (3.12) are our fundamental equations for the OCP on network reactors. Being a finite dimensional system of algebraic equations, they are easily solved by elementary means. (Observe that we are at this point assuming that $D_i(b)$ and $\nu_i(b)$ are constant on branches.) If $\nu_i(b) = 0$ but allow $D_i(x)$ to vary along b, it is possible to reparametrize b so as to make $D_i = 1$. For simplicity, we assume that D_i is already constant on branches.

In the remaining of this section, we rewrite the linear system of algebraic equations satisfied by $f(n)$ in a convenient form, highlighting a useful concept which we call *velocity-adjusted length*. In Chapter 4, we give several examples to illustrate how $f(n)$ is obtained explicitly.

3.7 Velocity-adjusted lengths

Equation (3.10) can be readily solved by elementary means. Let b be a branch attached to n having length $\ell(b)$, and $x \in [0, \ell(b)]$ the arc-length parameter along b. We choose the parametrization that orients b away from n (so that n corresponds to $x = 0$). Recall that

$F_{ij}(n,b) := f'_{ij}(n,b)$ represents the derivative at $x = 0$ of the restriction of f_{ij} to b. To be more explicit, we write $\mathbf{b} = (n,n')$. Then

$$f_{ij}(x) = f_{ij}(n) + \frac{1 - \exp\left\{-\frac{\nu_i(n,n')}{D_i(b)}x\right\}}{\nu_i(n,n')/D_i(b)} F_{ij}(n,b). \tag{3.13}$$

In particular,

$$F_{ij}(n,b) = F_{ij}(n,n') = \frac{f_{ij}(n') - f_{ij}(n)}{\tilde{\ell}_i(n,n')}, \tag{3.14}$$

where we have introduced the quantity

$$\tilde{\ell}_i(\mathbf{b}) = \tilde{\ell}_i(n,n') := \begin{cases} \frac{1-\exp\{-\ell(b)\nu_i(n,n')/D_i(b)\}}{\nu_i(n,n')/D_i(b)} & \text{if } \nu_i(n,n') \neq 0 \\ \\ \ell(b) & \text{if } \nu_i(n,n') = 0. \end{cases}$$

This positive quantity has physical dimension of length and it is continuous in $\nu_i(\mathbf{b})$, which is to say that $\tilde{\ell}_i(\mathbf{b}) \to \ell(b)$ when the advection velocity approaches 0. We refer to $\tilde{\ell}_i(\mathbf{b})$ as the (oriented) *velocity-adjusted length* of the branch b. Observe that $\tilde{\ell}(\mathbf{b})$ is always positive and depends on the orientation of the branch: if $\nu_i(\mathbf{b})$ is positive, transport of i in the direction of $\mathbf{b}$ is faster, and the adjusted length of b is less than $\ell(b)$ while transport of i in the direction of $\overline{\mathbf{b}}$ is slower, and the adjusted length of b is greater than $\ell(b)$. Strictly speaking, this adjusted length also depends on diffusivity. Large diffusivity negates the effect of velocity by making the quotient $\nu_i(\mathbf{b})/D_i(b)$ smaller in absolute value without changing its sign, while small diffusivity accentuates the velocity adjustment. Introducing the dimensionless quantity $s_i(\mathbf{b}) = \ell(b)\nu_i(\mathbf{b})/D_i(b)$, we can write the above relation in more transparent form:

$$\tilde{\ell}_i(\mathbf{b})/\ell(b) = \frac{1 - e^{-s_i(\mathbf{b})}}{s_i(\mathbf{b})}.$$

This function is positive, equals 1 at $s_i(\mathbf{b}) = 0$, decreases to 0 at the rate $1/s_i(\mathbf{b})$ as $s_i(\mathbf{b}) \to \infty$ and grows exponentially to $+\infty$ as $s_i(\mathbf{b}) \to -\infty$.

In our solution to the OCP, lengths of branches will always appear as $\tilde{\ell}_i(\mathbf{b})$ (or $\ell(b)$, when $\nu_i(\mathbf{b}) = 0$). The velocity-adjusted length is, thus, an effective length of branches resulting from a competition between velocity and diffusivity.

3.8 Solution to the output composition problem on network reactors

Given the observations of the previous section, the entries $f_{ij}(n)$ of the output composition matrix on internal nodes n are now obtained directly from Equations (3.10), (3.11), (3.12) and (3.13) as solution to an ordinary linear system of algebraic equations. To make this linear system more easily readable, it helps to introduce the following quantities. Let n be a node and $b = (n, n')$ a branch attached to n. Keep in mind that the notations (n, b) and (n, n') both represent an oriented branch $\mathbf{b}$ with initial node n. For each i we define

$$\xi_i(n,b) = \xi_i(n,n') := \frac{p(n,b)D_i(b)}{\tilde{\ell}_i(n,n')}, \quad \eta_i(n,b) = \eta_i(n,n') = \frac{\xi_i(n,n')}{\sum_{b'} \xi_i(n,b')}$$

where the sum in the denominator of $\eta_i(n,b)$ is over all branches b' attached to n. Then $\xi_i(n,b)$ has physical dimension length/time and $\eta_i(n,b)$ is dimensionless. Let $\eta(n,b) = \mathrm{diag}(\eta_1(n,b), \cdots, \eta_N(n,b))$, an $N \times N$ diagonal matrix. Finally, we introduce the dimensionless reaction coefficients

$$\tilde{K}_{ij}(n) := \frac{K_{ij}(n)}{\sum_{b'} \xi_i(n,b')}$$

where the sum is over all the branches b' connected to n.

We are now ready to write down the linear system for $f_{ij}(n)$. Suppose that the network contains $L+1$ nodes so that $n_1, n_2, \ldots, n_L$ are the internal nodes and $n_{L+1} = n_{\text{exit}}$ is the exit node. We define a matrix Λ of size $NL \times NL$, which we write in block-form, with blocks of size

$N \times N$, as follows. For each pair n_i, n_j of distinct internal nodes, the $N \times N$ block $\Lambda(n_i, n_j)$ of Λ at row i and column j is

$$\Lambda(n_i, n_j) = \begin{cases} \eta(n_i, n_j) & \text{if } i \neq j \\ \tilde{K}(n_i) - I & \text{if } i = j. \end{cases} \tag{3.15}$$

It is implicit in the above expression that $\tilde{K}(n_i) = 0$ if the interior node n_i is not active and $\eta(n_i, n_j) = 0$ if (n_i, n_j) (or its opposite) is not a branch of the network. Let

$$f = \begin{pmatrix} f(n_1) \\ \vdots \\ f(n_L) \end{pmatrix}, \quad \lambda = \begin{pmatrix} -\eta(n_1, n_{\text{exit}}) \\ \vdots \\ -\eta(n_L, n_{\text{exit}}) \end{pmatrix}.$$

These are $NL \times N$-sized matrices written in block-form. Then the output composition matrix is the solution to the linear system:

$$\Lambda f = \lambda. \tag{3.16}$$

Explicitly,

$$\begin{pmatrix} \tilde{K}(n_1) - I & \eta(n_1, n_2) & \cdots & \eta(n_1, n_L) \\ \eta(n_2, n_1) & \tilde{K}(n_2) - I & \cdots & \eta(n_2, n_L) \\ \vdots & \vdots & \ddots & \vdots \\ \eta(n_L, n_1) & \eta(n_L, n_2) & \cdots & \tilde{K}(n_L) - I \end{pmatrix} \begin{pmatrix} f(n_1) \\ f(n_2) \\ \vdots \\ f(n_L) \end{pmatrix} = - \begin{pmatrix} \eta(n_1, n_{\text{exit}}) \\ \eta(n_2, n_{\text{exit}}) \\ \vdots \\ \eta(n_L, n_{\text{exit}}) \end{pmatrix} \tag{3.17}$$

with blocks of size $N \times N$, where N is the number of gas species. This is our fundamental system of equations.

As an example, for the network diagram of Figure 3.1 (in which the interior nodes are $n_0, \ldots, n_4$), this system becomes (notice that $\eta(n_i, n_j) = I$ if there is a single branch issuing from n_i)

$$\begin{pmatrix} -I & I & 0 & 0 & 0 \\ 0 & -I & \eta(n_1,n_2) & \eta(n_1,n_3) & \eta(n_1,n_4) \\ 0 & \eta(n_2,n_1) & \tilde{K}(n_2)-I & 0 & \eta(n_2,n_4) \\ 0 & \eta(n_3,n_1) & 0 & \tilde{K}(n_3)-I & \eta(n_3,n_4) \\ 0 & \eta(n_4,n_1) & \eta(n_4,n_2) & \eta(n_4,n_3) & -I \end{pmatrix} \begin{pmatrix} f(n_0) \\ f(n_1) \\ f(n_2) \\ f(n_3) \\ f(n_4) \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \eta(n_4,n_5) \end{pmatrix}. \tag{3.18}$$

Before exploring Equation (3.17) further, it is natural to ask whether the coefficient matrix Λ is indeed invertible. This is to be expected since this linear system arises from a boundary-value problem for a system of partial differential equations whose solutions are uniquely determined. Nevertheless, it is reassuring to be able to ascertain solvability independently by elementary means under very general assumptions. This point is discussed in section 5.4.2.

4. Network Reactor Examples

In all the examples to be considered we make the following convenient but natural assumptions: $p(n,b) = 1/\deg(n)$ (all pipes have the same cross-section), $D_i(b) = D$ does not depend on the chemical species and the branch, and $\nu_i(\mathbf{b}) = \nu(\mathbf{b})$ is the same for all species but may depend on the branch. Thus, for a choice $\mathbf{b} = (n, n')$ of orientation for b,

$$\tilde{\ell}(\mathbf{b}) = \frac{1 - e^{-\nu(\mathbf{b})\ell(b)/D}}{\nu(\mathbf{b})/D}, \quad \xi(n,n') = \frac{D/\tilde{\ell}(\mathbf{b})}{\deg(n)}, \quad \eta(n,n') = \frac{\xi(n,n')}{\sum_{n''}\xi(n,n'')}$$

do not depend on i. The sum in the denominator of $\eta(n,n')$ is over all branches (n,n'') attached to n. Since $\eta(n,n')$ can now be viewed as a scalar, we write $\eta(n,n')I$ for the corresponding $N \times N$ matrix, where I is the identity matrix. When the branches are indexed, b_i, it is useful to write $\tilde{\ell}_i := \tilde{\ell}(\mathbf{b}_i)$ and $\tilde{\ell}_{\bar{i}} := \tilde{\ell}(\overline{\mathbf{b}}_i)$, where $\mathbf{b}_i$ has the orientation indicated by an arrow in the network diagram of each example.

4.1 Segment reactor with 1 active node

For the example of Figure 4.1, we have

$$\tilde{\ell}_0 := \tilde{\ell}(\overline{b}_0) = \frac{e^{\nu(b_0)\ell(b_0)/D} - 1}{\nu(b_0)/D}, \quad \tilde{\ell}_1 := \tilde{\ell}(b_1) = \frac{1 - e^{-\nu(b_1)\ell(b_1)/D}}{\nu(b_1)/D},$$

so that

$$\eta(n_1,n_0) = \frac{\tilde{\ell}_1}{\tilde{\ell}_0 + \tilde{\ell}_1}, \quad \eta(n_1,n_2) = \frac{\tilde{\ell}_0}{\tilde{\ell}_0 + \tilde{\ell}_1}, \quad \tilde{K} := \tilde{K}(n_1) = \frac{K(n_1)}{\frac{1}{2}D\left(\frac{1}{\tilde{\ell}_0} + \frac{1}{\tilde{\ell}_1}\right)} = \frac{2\tilde{\ell}_0\tilde{\ell}_1}{\tilde{\ell}_0 + \tilde{\ell}_1}\frac{K(n_1)}{D}.$$

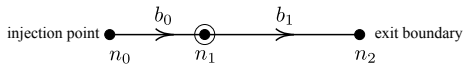


Figure 4.1.: A simple network reactor with a single active node.

Therefore,

$$\begin{pmatrix} -I & I \\ \frac{\tilde{\ell}_1}{\tilde{\ell}_0+\tilde{\ell}_1}I & \tilde{K}-I \end{pmatrix}\begin{pmatrix} f(n_0) \\ f(n_1) \end{pmatrix} = -\begin{pmatrix} 0 \\ \frac{\tilde{\ell}_0}{\tilde{\ell}_0+\tilde{\ell}_1}I \end{pmatrix}.$$

This system is easily solved. The result is

$$f(n_0) = f(n_1) = \left(I - \frac{2\tilde{\ell}_1}{D}K\right)^{-1}. \tag{4.1}$$

As could have been expected, $\tilde{\ell}_0$ does not appear in the solution. Had we assumed $\ell_0 = 0$, however, the factor of 2 in front of $\tilde{\ell}_1$ would be 1 instead, since the degree of n_1 would go from 2 to 1.

Let us explore this solution in some detail. Recall that $f_{ij}(n_0)$ is the fraction of j in the output given that a unit pulse of i is initially injected at node n_0. Suppose there are only two chemical species, denoted $1, 2$ with reactions $1 \rightleftharpoons 2$ so that

$$K = \begin{pmatrix} -k_+ & k_+ \\ k_- & -k_- \end{pmatrix}.$$

(k_+ is the coefficient of $1 \to 2$ and k_- is the coefficient for the reverse reaction $2 \to 1$.) The inverse matrix in Equation (4.1) is easily found:

$$f(n_0) = \frac{1}{1 + \frac{2\tilde{\ell}_1}{D}(k_- + k_+)}\begin{pmatrix} 1 + \frac{2\tilde{\ell}_1 k_-}{D} & \frac{2\tilde{\ell}_1 k_+}{D} \\ \frac{2\tilde{\ell}_1 k_-}{D} & 1 + \frac{2\tilde{\ell}_1 k_+}{D} \end{pmatrix}.$$

Thus, for example, the fraction of species 2 in the output given that a pulse containing only species 1 was initially injected at n_0 is

$$f_{12}(n_0) = \frac{\frac{2\tilde{\ell}_1}{D}k_+}{1 + \frac{2\tilde{\ell}_1}{D}(k_- + k_+)}.$$

From $f(n_0)$ we can determine the output composition for any composition of substances initially injected into the reactor. Suppose that the initial quantities of 1 and 2 are given by the vector $\alpha = (\alpha_1, \alpha_2)$. Then the output amounts are given by the vector $\beta = (\beta_1, \beta_2)$ such that $\beta = \alpha f(n_0)$ (matrix multiplication). In particular,

$$\frac{\beta_2}{\beta_1} = \left[k_+ + \frac{D}{2\tilde{\ell}_1}\frac{\alpha_2}{\alpha_1 + \alpha_2}\right] \Big/ \left[k_- + \frac{D}{2\tilde{\ell}_1}\frac{\alpha_1}{\alpha_1 + \alpha_2}\right].$$

Observe the effect of varying the transport coefficient $D/\tilde{\ell}_1$. If this coefficient is small, the above ratio is approximately

$$\frac{\beta_2}{\beta_1} \approx \frac{k_+}{k_-}.$$

This is the equilibrium value for a closed reactor. Such situation arises when the diffusion coefficient is very small or b_1 is very long. Equivalently, this approximation holds when the advection velocity $\nu(b_1)$ is negative with large absolute value. The quantity $2\tilde{\ell}_1/D$ may be viewed as a measure of the time spent at the active node. It has physical dimension time/length rather than time for the same reason that k does not have dimension 1/time when the active region reduces to a point. (The actual time spent at a single point is 0).

It is also interesting to observe in this example that, independently of the transport characteristics,

$$\frac{\text{output fraction of } 2 \text{ given initial unit pulse of } 1}{\text{output fraction of } 1 \text{ given initial unit pulse of } 2} = \frac{f_{12}(n_0)}{f_{21}(n_0)} = \frac{k_+}{k_-} = \text{ equilibrium ratio.}$$

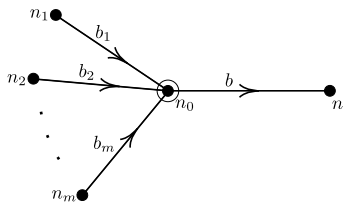


Figure 4.2.: Reactants injected at different initial nodes.

We refer to [24] and [25] for the significance of a reciprocity relation of similar kind but in a different setting.

4.2 Distributed input

A simple variant of the previous example helps to illustrate the situation in which a mixture is injected over several nodes. Consider the reactor of Figure 4.2. Suppose m different gas species. A unit pulse containing fractions $\alpha_1, \ldots, \alpha_m$ of each species $1, \ldots, m$ is injected so that species i is injected at node n_i. Then the fraction of j in the output is

$$f_j = \alpha_1 f_{1j}(n_1) + \cdots + \alpha_m f_{mj}(n_m).$$

It is easily shown (similarly to the first example) that $f(n_1) = \cdots = f(n_m) = f(n_0)$, independently of the lengths and advection velocities of branches $b_1, \ldots, b_m$.

The matrix $f(n_0)$ is also easily obtained:

$$f(n_0) = \left(I - \frac{(m+1)\tilde{\ell}(b)}{D} K \right)^{-1}.$$

So, in fact, $f_j = \alpha_1 f_{1j}(n_0) + \cdots + \alpha_m f_{mj}(n_0)$. This means that, in this specific situation (where the injection nodes are connected to the exit node through a single path passing through the one active node n_0), the process is equivalent to injecting the whole pulse mixture at once at n_0.

Incidentally, it is not difficult to show directly that a square matrix of the form $I - \mu K$, where $\mu \geq 0$ and $K = (K_{ij})$ is such that $K_{ij} \geq 0$ for $i \neq j$, $K_{ii} = -\sum_{j \neq i} K_{ij}$, is always invertible. For the special case $m = 2$ and $K = \begin{pmatrix} -k_+ & k_+ \\ k_- & -k_- \end{pmatrix}$, we have

$$f(n_0) = \frac{1}{1 + \frac{3\tilde{\ell}(b)}{D}(k_- + k_+)} \begin{pmatrix} 1 + \frac{3\tilde{\ell}(b)}{D}k_- & \frac{3\tilde{\ell}(b)}{D}k_+ \\ \frac{3\tilde{\ell}(b)}{D}k_- & 1 + \frac{3\tilde{\ell}(b)}{D}k_+ \end{pmatrix}$$

4.3 A segment reactor with one active node and a bypass.

The network reactor for this example is shown in Figure 4.3. One easily finds

$$\eta(n_0, n_1) = \frac{\tilde{\ell}_2}{\tilde{\ell}_0 + \tilde{\ell}_2}, \quad \eta(n_1, n_0) = \frac{\tilde{\ell}_1}{\tilde{\ell}_{\overline{0}} + \tilde{\ell}_1}, \quad \eta(n_0, n_2) = \frac{\tilde{\ell}_0}{\tilde{\ell}_0 + \tilde{\ell}_2}, \quad \eta(n_1, n_2) = \frac{\tilde{\ell}_{\overline{0}}}{\tilde{\ell}_{\overline{0}} + \tilde{\ell}_1}$$

so that

$$\begin{pmatrix} -I & \frac{\tilde{\ell}_2}{\tilde{\ell}_0 + \tilde{\ell}_2} I \\ \frac{\tilde{\ell}_1}{\tilde{\ell}_{\overline{0}} + \tilde{\ell}_1} I & \tilde{K} - I \end{pmatrix} \begin{pmatrix} f(n_0) \\ f(n_1) \end{pmatrix} = - \begin{pmatrix} \frac{\tilde{\ell}_0}{\tilde{\ell}_0 + \tilde{\ell}_2} I \\ \frac{\tilde{\ell}_{\overline{0}}}{\tilde{\ell}_{\overline{0}} + \tilde{\ell}_1} I \end{pmatrix}.$$

The first equation in this system may be written as

$$f(n_0) = \frac{\tilde{\ell}_2}{\tilde{\ell}_0 + \tilde{\ell}_2} f(n_1) + \frac{\tilde{\ell}_0}{\tilde{\ell}_0 + \tilde{\ell}_2} I, \tag{4.2}$$

which has a natural interpretation. If we regard the velocity-adjusted length $\tilde{\ell}(b)$ as the resistance to crossing branch b, then $1/\tilde{\ell}(b)$ may be interpreted as a conductivity. Transport from n_0 to n_1 and from n_0 to n_2 is then distributed in proportion to the relative conductivity of the two channels. Writing these relative conductivities as a_1 and a_2 ($a_1 + a_2 = 1$) then Equation (4.2) becomes

$$f_{ij}(n_0) = a_1 f_{ij}(n_1) + a_2 f_{ij}(n_2)$$

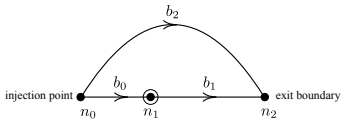


Figure 4.3.: A simple network reactor with a single active node and a bypass.

for each (i, j). Expressed in words, the molar fraction of j in the output mixture given that a unit pulse of i is injected at n_0 equals the weighted average of these molar fractions for the injection points n_1 ($f_{ij}(n_1)$) and n_2 ($f_{ij}(n_2) = \delta_{ij}$), with weights given by the channels' relative conductivity.

It remains to obtain $f(n_1)$, the output composition when the injection is at the active node. This is easily found to be

$$f(n_1) = \left(I - \frac{\left(\tilde{\ell}_{\bar{0}} + \tilde{\ell}_1\right)\left(\tilde{\ell}_0 + \tilde{\ell}_2\right)}{\tilde{\ell}_{\bar{0}}\tilde{\ell}_0 + \tilde{\ell}_{\bar{0}}\tilde{\ell}_2 + \tilde{\ell}_1\tilde{\ell}_0} \tilde{K} \right)^{-1} = (I - \alpha K)^{-1}, \tag{4.3}$$

where

$$\alpha := \frac{2\tilde{\ell}_{\bar{0}}\tilde{\ell}_1\left(\tilde{\ell}_0 + \tilde{\ell}_2\right)}{D\left(\tilde{\ell}_{\bar{0}}\tilde{\ell}_0 + \tilde{\ell}_{\bar{0}}\tilde{\ell}_2 + \tilde{\ell}_1\tilde{\ell}_0\right)}, \quad \tilde{K} := \frac{2\tilde{\ell}_{\bar{0}}\tilde{\ell}_1}{D\left(\tilde{\ell}_{\bar{0}} + \tilde{\ell}_1\right)} K.$$

In the special case of only two species,

$$f(n_1) = \frac{1}{1 + \alpha(k_- + k_+)} \begin{pmatrix} 1 + \alpha k_- & \alpha k_+ \\ \alpha k_- & 1 + \alpha k_+ \end{pmatrix}.$$

We highlight again the reciprocal relation

$$\frac{f_{12}(n_0)}{f_{21}(n_0)} = \frac{f_{12}(n_1)}{f_{21}(n_1)} = \frac{k_+}{k_-}.$$

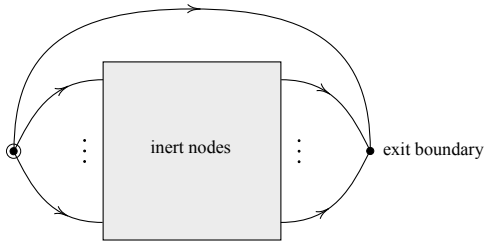


Figure 4.4.: The general network reactor with a single active node. The box contains a network of L inert nodes connected to the single active node, n_0, on the left and the exit node, n_{exit}, on the right.

4.4 The general single active node network

A diagram for the general single active node network is shown in Figure 4.4. The system for f is

$$\begin{pmatrix} \tilde{K} - I & \eta(n_0, n_1)I & \cdots & \eta(n_0, n_L)I \\ \eta(n_1, n_0)I & -I & \cdots & \eta(n_1, n_L)I \\ \vdots & \vdots & \ddots & \vdots \\ \eta(n_L, n_0)I & \eta(n_L, n_1)I & \cdots & -I \end{pmatrix} \begin{pmatrix} f(n_0) \\ f(n_1) \\ \vdots \\ f(n_L) \end{pmatrix} = - \begin{pmatrix} \eta(n_0, n_{\text{exit}}) \\ \eta(n_1, n_{\text{exit}}) \\ \vdots \\ \eta(n_L, n_{\text{exit}}) \end{pmatrix}$$

It is implicit in this expression that $\eta(n, n') = 0$ when n and n' are not connected by a branch. Note that all the block entries of the coefficient matrix commute with each other.

By Cramer's rule for the inverse of the coefficient matrix Λ, we obtain

$$f(n_{\text{init}}) = (\alpha(n_{\text{init}})I + \beta(n_{\text{init}})K)\,(\delta(n_{\text{init}})I + \gamma(n_{\text{init}})K)^{-1} \tag{4.4}$$

where $\alpha, \beta, \gamma, \delta$ are polynomial functions of the velocity-adjusted branch lengths that take into account the geometry and topology of the network reactor as well as the point of injection n_{init}. From the property that the sum of each row of $f(n_{\text{init}})$ is 1 and the sum of each row of K is 0, it can be shown that $\alpha = \delta$. So $f(n_{\text{init}})$ can be written as

$$f(n_{\text{init}}) = (I + \xi(n_{\text{init}})K)(I + \eta(n_{\text{init}})K)^{-1}, \tag{4.5}$$

where $\xi = \beta/\alpha$ and $\eta = \gamma/\alpha$. From this remark, it is not difficult to conclude that if $c = (c_1, \ldots, c_N)$ is the vector of equilibrium concentrations for the reaction matrix K for a closed reactor, so that $cK = 0$, then

$$cf(n_{\text{init}}) = c.$$

This means that if the proportions of the substances in the input composition are the same as for the equilibrium concentrations in a closed reactor, then the output composition is the same as the input composition.

For a concrete example, consider the reaction system $1 \rightleftharpoons 2$ with matrix of reaction coefficients

$$K = \begin{pmatrix} -k_+ & k_+ \\ k_- & -k_- \end{pmatrix},$$

we obtain (using $\alpha = \alpha(n_{\text{init}})$, etc.)

$$f(n_{\text{init}}) = \frac{1}{\alpha(\delta - \gamma(k_- + k_+))} \begin{pmatrix} \alpha\delta - \alpha\gamma k_- - \beta\delta k_+ & (\beta\delta - \alpha\gamma)k_+ \\ (\beta\delta - \alpha\gamma)k_- & \alpha\delta - \alpha\gamma k_+ - \beta\delta k_- \end{pmatrix}$$

$$= \frac{1}{1 - \eta(k_+ + k_-)} \begin{pmatrix} 1 - \eta k_- - \xi k_+ & (\xi - \eta)k_+ \\ (\xi - \eta)k_- & 1 - \xi k_- - \eta k_+ \end{pmatrix}$$

In particular, for a general network reactor with a single active node and one pair of reversible reactions involving only two substances, the reciprocal relation $f_{12}(n_{\text{init}})/f_{21}(n_{\text{init}}) = k_+/k_-$ holds.

4.5 A linear network reactor with two active nodes

Let us consider the linear network reactor with two active nodes shown in Figure 4.5.

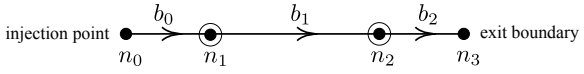


Figure 4.5.: Linear network with two active nodes.

Using the same notational conventions as before, we have the linear system

$$\begin{pmatrix} -I & I & 0 \\ \frac{\tilde{\ell}_1}{\tilde{\ell}_{\overline{0}}+\tilde{\ell}_1} I & \tilde{K}_1 - I & \frac{\tilde{\ell}_{\overline{0}}}{\tilde{\ell}_{\overline{0}}+\tilde{\ell}_1} I \\ 0 & \frac{\tilde{\ell}_2}{\tilde{\ell}_{\overline{1}}+\tilde{\ell}_2} I & \tilde{K}_2 - I \end{pmatrix} \begin{pmatrix} f(n_0) \\ f(n_1) \\ f(n_2) \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ \frac{\tilde{\ell}_{\overline{1}}}{\tilde{\ell}_{\overline{1}}+\tilde{\ell}_2} I \end{pmatrix}$$

The solution may be written in matrix form as

$$f(n_2) = \left[I - \frac{2}{D} \left(\frac{\tilde{\ell}_1 \left(\tilde{\ell}_{\overline{1}} + \tilde{\ell}_2 \right)}{\tilde{\ell}_{\overline{1}}} K_1 + \tilde{\ell}_2 K_2 \right) + \frac{4}{D^2} \tilde{\ell}_1 \tilde{\ell}_2 K_1 K_2 \right]^{-1} \left(I - \frac{2\tilde{\ell}_1}{D} K_1 \right)$$

$$f(n_1) = \left(I - \frac{2\tilde{\ell}_1}{D} K_1 \right)^{-1} f(n_2)$$

$$f(n_0) = f(n_1).$$

In particular,

$$f(n_0) = \left(I - \frac{2\tilde{\ell}_1}{D} K_1 \right)^{-1} \left[I - \frac{2}{D} \left(\frac{\tilde{\ell}_1 \left(\tilde{\ell}_{\overline{1}} + \tilde{\ell}_2 \right)}{\tilde{\ell}_{\overline{1}}} K_1 + \tilde{\ell}_2 K_2 \right) + \frac{4}{D^2} \tilde{\ell}_1 \tilde{\ell}_2 K_1 K_2 \right]^{-1} \left(I - \frac{2\tilde{\ell}_1}{D} K_1 \right). \tag{4.6}$$

Note that the first and third factors in the above expression for $f(n_0)$ cancel out when K_1 and K_2 commute. This is the case, for example, when the two nodes implement the same reactions, that is $K_1 = K_2$, or when they implement reactions involving non-intersecting sets of species, making the reactions at n_1 and n_2 independent. In the latter case, K_1 and K_2 have diagonal block form

$$K_1 = \left(\begin{array}{c|c} L_1 & \\ \hline & O_2 \end{array} \right), \quad K_2 = \left(\begin{array}{c|c} O_1 & \\ \hline & L_2 \end{array} \right),$$

where O_1, O_2 are the zero square matrices, L_1, L_2 are rate constant matrices, and O_i and L_i have the same size for $i = 1, 2$. From this form it also follows that $K_1 K_2 = 0$. Therefore, in this case, we have

$$f(n_0) = \left[I - \frac{2}{D} \left(\begin{array}{c|c} \frac{\tilde{\ell}_1(\tilde{\ell}_{\bar{1}} + \tilde{\ell}_2)}{\tilde{\ell}_{\bar{1}}} L_1 & \\ \hline & \tilde{\ell}_2 L_2 \end{array} \right) \right]^{-1}.$$

Let us write the solution more explicitly in a couple of special cases. First suppose that the only reactions are $1 \rightleftharpoons 2$, and they have the same constants at n_1 and n_2. That is,

$$K_1 = K_2 = \begin{pmatrix} -k_+ & k_+ \\ k_- & -k_- \end{pmatrix}.$$

Then

$$f(n_0) = \frac{1}{1 + \alpha(k_- + k_+) + \beta(k_- + k_+)^2} \begin{pmatrix} 1 + \alpha k_- + \beta k_-(k_- + k_+) & \alpha k_+ + \beta k_+(k_- + k_+) \\ \alpha k_- + \beta k_-(k_- + k_+) & 1 + \alpha k_+ + \beta k_+(k_- + k_+) \end{pmatrix}$$

where

$$\alpha = \frac{2}{D} \frac{\tilde{\ell}_1 \tilde{\ell}_2 + \tilde{\ell}_1 \tilde{\ell}_{\bar{1}} + \tilde{\ell}_2 \tilde{\ell}_{\bar{1}}}{\tilde{\ell}_1}, \quad \beta = \frac{4}{D^2} \tilde{\ell}_1 \tilde{\ell}_2.$$

Thus, for example, if 2 is initially injected at n_0, the molar fraction of 1 in the output composition is

$$f_{21}(n_0) = \frac{\alpha k_- + \beta k_-(k_- + k_+)}{1 + \alpha(k_- + k_+) + \beta(k_- + k_+)^2}.$$

Here again, as in the first example, we observe the symmetric relation:

$$\frac{f_{12}(n_0)}{f_{21}(n_0)} = \frac{\alpha k_+ + \beta k_+(k_- + k_+)}{\alpha k_- + \beta k_-(k_- + k_+)} = \frac{k_+}{k_-}.$$

As a second instance of the same network, let us suppose that there are 3 chemical species, and that n_1 implements the reaction $1 \rightleftharpoons 2$ and n_2 implements the reaction $2 \rightleftharpoons 3$. The reaction matrices for this system are

$$K_1 = \begin{pmatrix} -k_+ & k_+ & 0 \\ k_- & -k_- & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad K_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -k'_+ & k'_+ \\ 0 & k'_- & -k'_- \end{pmatrix}.$$

Let us define the constants

$$\alpha = \frac{2}{D}\tilde{\ell}_1, \quad \beta = \frac{2}{D}\frac{\tilde{\ell}_1\left(\tilde{\ell}_2 + \tilde{\ell}_{\overline{1}}\right)}{\tilde{\ell}_{\overline{1}}}, \quad \gamma = \frac{2}{D}\tilde{\ell}_2, \quad \delta = \frac{4}{D^2}\tilde{\ell}_1\tilde{\ell}_2.$$

Then, using Equation (4.6),

$$f(n_0) = \begin{pmatrix} 1+\alpha k_+ & -\alpha k_+ & 0 \\ -\alpha k_- & 1+\alpha k_- & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1+\beta k_+ & -\beta k_+ + \delta k_+ k'_+ & \delta k_+ k'_+ \\ -\beta k_- & 1+\beta k_- + \gamma k'_+ + \delta k_- k'_+ & -\gamma k'_+ - \delta k_- k'_+ \\ 0 & -\gamma k'_- & 1+\gamma k'_- \end{pmatrix}^{-1} \begin{pmatrix} 1+\alpha k_+ & -\alpha k_+ & 0 \\ -\alpha k_- & 1+\alpha k_- & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

From the evaluation of this matrix, we can find, for example, $f_{31}(n_0)$, the fraction represented by species 1 in the output given an initial unit pulse of 3 injected at n_0:

$$f_{31}(n_0) = \frac{\delta(\tilde{\ell}_2/\tilde{\ell}_{\overline{1}})k_-k'_-}{1+\beta(k_-+k_+)+\gamma(k'_-+k'_+)+\beta\gamma(k_-k'_-+k'_-k_++k_+k'_+)+\delta k_-k'_++2\beta\delta k_-k_+k'_++2\beta\gamma\delta k_-k'_-k_+k'_+}.$$

4.6 A network reactor with parallel active nodes

To the network of Figure 4.6 is associated the system

$$\begin{pmatrix} -I & \eta(n_0,n_1)I & \eta(n_0,n_2)I \\ \eta(n_1,n_0)I & \tilde{K}_1 - I & 0 \\ \eta(n_2,n_0)I & 0 & \tilde{K}_2 - I \end{pmatrix} \begin{pmatrix} f(n_0) \\ f(n_1) \\ f(n_2) \end{pmatrix} = - \begin{pmatrix} 0 \\ \eta(n_1,n_3)I \\ \eta(n_2,n_3)I \end{pmatrix}$$

where

$$\eta(n_0,n_1) = 1-\eta(n_0,n_2) = \frac{\tilde{\ell}_3}{\tilde{\ell}_1+\tilde{\ell}_3},\ \ \eta(n_1,n_0) = 1-\eta(n_1,n_3) = \frac{\tilde{\ell}_2}{\tilde{\ell}_2+\tilde{\ell}_{\overline{1}}},\ \ \eta(n_2,n_0) = 1-\eta(n_2,n_3) = \frac{\tilde{\ell}_4}{\tilde{\ell}_4+\tilde{\ell}_{\overline{3}}}$$

and

$$\tilde{K}_1 = \frac{2}{D}\frac{\tilde{\ell}_2\tilde{\ell}_{\overline{1}}}{\tilde{\ell}_2+\tilde{\ell}_{\overline{1}}}K_1,\ \ \tilde{K}_2 = \frac{2}{D}\frac{\tilde{\ell}_4\tilde{\ell}_{\overline{3}}}{\tilde{\ell}_4+\tilde{\ell}_{\overline{3}}}K_2.$$

First note that

$$f(n_0) = \frac{\tilde{\ell}_3}{\tilde{\ell}_1+\tilde{\ell}_3}f(n_1) + \frac{\tilde{\ell}_1}{\tilde{\ell}_1+\tilde{\ell}_3}f(n_2).$$

That is, the output composition matrix for the injection point n_0 is the weighted average of those with injection points at the active nodes. (The longer the velocity-adjusted length of a branch, the lesser the weight of that branch among the possible channels leading to active nodes.)

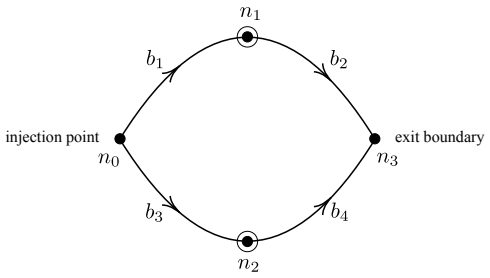


Figure 4.6.: Two active nodes in parallel.

To obtain the latter, it helps to set the following notation.

$$a_1 = \frac{2}{D}\frac{\tilde{\ell}_{\overline{1}}\tilde{\ell}_2(\tilde{\ell}_1+\tilde{\ell}_3)}{\tilde{\ell}_1\tilde{\ell}_{\overline{1}}+\tilde{\ell}_{\overline{1}}\tilde{\ell}_3+\tilde{\ell}_1\tilde{\ell}_2},\ \ a_2 = \frac{\tilde{\ell}_1\tilde{\ell}_3(\tilde{\ell}_{\overline{1}}+\tilde{\ell}_2)}{(\tilde{\ell}_1+\tilde{\ell}_3)(\tilde{\ell}_1\tilde{\ell}_{\overline{1}}+\tilde{\ell}_{\overline{1}}\tilde{\ell}_3+\tilde{\ell}_1\tilde{\ell}_2)},\ \ a_3 = \frac{\tilde{\ell}_3\tilde{\ell}_4}{\tilde{\ell}_3\tilde{\ell}_4+\tilde{\ell}_1\tilde{\ell}_{\overline{3}}+\tilde{\ell}_3\tilde{\ell}_{\overline{3}}},\ \ a_4 = \frac{2}{D}\frac{\tilde{\ell}_{\overline{3}}\tilde{\ell}_4(\tilde{\ell}_1+\tilde{\ell}_3)}{\tilde{\ell}_3\tilde{\ell}_4+\tilde{\ell}_1\tilde{\ell}_{\overline{3}}+\tilde{\ell}_3\tilde{\ell}_{\overline{3}}}$$

and

$$b_1 = \frac{\tilde{\ell}_{\overline{1}}(\tilde{\ell}_1+\tilde{\ell}_3)}{\tilde{\ell}_1\tilde{\ell}_{\overline{1}}+\tilde{\ell}_{\overline{1}}\tilde{\ell}_3+\tilde{\ell}_1\tilde{\ell}_2},\ \ b_2 = \frac{\tilde{\ell}_{\overline{3}}(\tilde{\ell}_1+\tilde{\ell}_3)}{\tilde{\ell}_3\tilde{\ell}_4+\tilde{\ell}_1\tilde{\ell}_{\overline{3}}+\tilde{\ell}_3\tilde{\ell}_{\overline{3}}}.$$

Then the matrices $f(n_1), f(n_2)$ satisfy the system

$$(I-a_1K_1)f(n_1) - a_2f(n_2) = b_1,\ \ -a_3f(n_1) + (I-a_4K_2)f(n_2) = b_2.$$

Among the many possibilities one can explore, let us only write the solution for $K_1 = K_2 = K$, $\ell_i = \ell$ and $\nu(b_i) = 0$ for $i = 1, 2, 3, 4$. Then $\tilde{\ell}_i = \tilde{\ell}_{\bar{i}} = \ell$.

$$\begin{pmatrix} f(n_1) \\ \\ f(n_2) \end{pmatrix} = \left[\left(I - \frac{4\ell}{3D}K\right)^2 - \frac{1}{9}I\right]^{-1} \begin{pmatrix} I - \frac{4\ell}{3D}K & \frac{1}{3}I \\ \\ \frac{1}{3}I & I - \frac{4\ell}{3D}K \end{pmatrix} \begin{pmatrix} \frac{2}{3} \\ \\ \frac{2}{3} \end{pmatrix}.$$

This is easily solved:

$$f(n_1) = f(n_2) = \left(I - \frac{2\ell}{D}K\right)^{-1}.$$

For the simple reaction $1 \rightleftharpoons 2$, for which $K = \begin{pmatrix} -k_+ & k_+ \\ \\ k_- & -k_- \end{pmatrix}$,

$$f(n_1) = f(n_2) = \frac{1}{1 + \frac{2\ell}{D}(k_- + k_+)} \begin{pmatrix} 1 + \frac{2\ell}{D}k_- & \frac{2\ell}{D}k_+ \\ \\ \frac{2\ell}{D}k_- & 1 + \frac{2\ell}{D}k_+ \end{pmatrix}.$$

Once again we see the reciprocity relation

$$\frac{f_{12}(n_0)}{f_{21}(n_0)} = \frac{f_{12}(n_1)}{f_{21}(n_1)} = \frac{f_{12}(n_2)}{f_{21}(n_2)} = \frac{k_+}{k_-}.$$

4.7 Adsorption

In this thesis we do not yet develop systematically reaction mechanisms involving adsorption at active regions, but we illustrate how such reactions may be handled based on the methods considered so far. By adsorption we have in mind reactions of the form $A + Z \rightleftharpoons AZ$, where Z is a catalyst that remains, together with the complex AZ, at the active region while A migrates freely through the reactor via diffusion and advection.

Under the assumption that the amount and characteristics of Z at a node remain essentially unchanged over the time span of the process, we may replace the second order reaction with

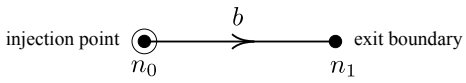


Figure 4.7.: A simple network reactor to illustrate adsorption reactions.

the first order $A \rightleftharpoons AZ$. It is interesting then to investigate whether introducing AZ as a new species having zero advection velocity and very small diffusivity D, and then passing to the limit as D approaches 0, gives meaningful results. We explore this possibility with one example and leave a systematic treatment for another study.

Let us consider the system of reactions

$$A \rightleftharpoons AZ \rightleftharpoons BZ \rightleftharpoons B$$

where A and B are gases, subject to adsorption and disorption, and AZ and BZ are catalytic intermediates. A and B will be called *migrating* and AZ and BZ *static*. For simplicity of notation, we use $1, 2, 3, 4$ for A, AZ, BZ, B, respectively. For the reactor we take the (simplest) example shown in Figure 4.7. See [10] for a justification of this mechanism.

We suppose that 1 and 4 have diffusivity D and advection velocity $\nu = \nu(b)$ while 2 and 3 have (small) diffusivity D', which will be taken to 0 at the end of the calculation, and zero advection velocity. Let us indicated the length of b by ℓ, its velocity-adjusted length by $\tilde{\ell}$ and the reaction coefficients by

$$1 \underset{\mathrm{k}_-^{(1)}}{\overset{\mathrm{k}_+^{(1)}}{\rightleftharpoons}} 2 \underset{\mathrm{k}_-}{\overset{\mathrm{k}_+}{\rightleftharpoons}} 3 \underset{\mathrm{k}_-^{(2)}}{\overset{\mathrm{k}_+^{(2)}}{\rightleftharpoons}} 4.$$

The output composition matrix is then the 4×4-matrix

$$f(n_0) = \left(I - \tilde{K}\right)^{-1}$$

where

$$\tilde{K}_{ij} = \begin{cases} \frac{\tilde{\ell}}{D} K_{ij} & \text{if } i = 1,4 \\ \\ \frac{\ell}{D'} K_{ij} & \text{if } i = 2,3 \end{cases} \quad \text{or} \quad \tilde{K} = \begin{pmatrix} -\frac{\tilde{\ell}}{D} k_+^{(1)} & \frac{\tilde{\ell}}{D} k_+^{(1)} & 0 & 0 \\ \\ \frac{\ell}{D'} k_-^{(1)} & -\frac{\ell}{D'} \left(k_-^{(1)} + k_+ \right) & \frac{\ell}{D'} k_+ & 0 \\ \\ 0 & \frac{\ell}{D'} k_- & -\frac{\ell}{D'} \left(k_- + k_+^{(2)} \right) & \frac{\ell}{D'} k_+^{(2)} \\ \\ 0 & 0 & \frac{\tilde{\ell}}{D} k_-^{(2)} & -\frac{\tilde{\ell}}{D} k_-^{(2)} \end{pmatrix}.$$

In the limit as D' approaches 0 we obtain

$$f(n_0) = \begin{pmatrix} \frac{k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right) + k_+^{(2)} \left(k_+ + k_-^{(1)}\right)}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} & 0 & 0 & \frac{\frac{\tilde{\ell}}{D} k_+ k_+^{(1)} k_+^{(2)}}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} \\ \\ \frac{k_-^{(1)} \left[k_+^{(2)} + k_- \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)\right]}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} & 0 & 0 & \frac{k_- k_+ \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right)}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} \\ \\ \frac{k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} & 0 & 0 & \frac{k_+^{(2)} \left[k_-^{(1)} + k_+ \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right)\right]}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} \\ \\ \frac{\frac{\tilde{\ell}}{D} k_- k_-^{(1)} k_-^{(2)}}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} & 0 & 0 & \frac{k_- k_-^{(1)} + k_+^{(2)} \left[k_-^{(1)} + k_+ \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right)\right]}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)} \end{pmatrix}. \tag{4.7}$$

As was to be expected, $f_{ij}(n_0) = 0$ whenever $j = 2, 3$. This is because the species AZ and BZ should remain at n_0 and not migrate to the exit. It is interesting to compare the fraction of B in the output given a unit pulse of A injected at n_0 and the corresponding value for the overall reaction $A \rightleftharpoons B$. The former is

$$f_{14}(n_0) = \frac{\frac{\tilde{\ell}}{D} k_+ k_+^{(1)} k_+^{(2)}}{k_+^{(2)} \left[k_-^{(1)} + \left(1 + \frac{\tilde{\ell}}{D} k_+^{(1)}\right) k_+\right] + k_- k_-^{(1)} \left(1 + \frac{\tilde{\ell}}{D} k_-^{(2)}\right)}.$$

For the latter, let us indicate the reaction coefficients by $K_\pm$. Then the fraction of B in the output given that a unit pulse of A was injected at n_0 is

$$f_{AB}(n_0) = \frac{\frac{\tilde{\ell}}{D} K_+}{1 + \frac{\tilde{\ell}}{D} \left(K_- + K_+\right)}.$$

Note that, if $k_-^{(1)} = k_-^{(2)} = k_+^{(2)} = k_+^{(1)}$ is taken to be very large so that adsorption/desorption are very fast reactions, then $f_{14}(n_0)$ reduces to $f_{AB}(n_0)$ for $K_\pm = k_\pm$.

Finally, it is again notable from Equation (4.7) the reciprocal relation

$$\frac{f_{14}(n_0)}{f_{41}(n_0)} = \frac{k_+ k_+^{(1)} k_+^{(2)}}{k_- k_-^{(1)} k_-^{(2)}},$$

which does not depend on transport coefficients.

4.7.1 Numerical investigation of the network of Figure 3.1

Examining the system of equations (3.18), we immediately see that $f(n_0) = f(n_1)$. This means that injecting the initial pulse at n_0 has exactly the same effect as injecting it at n_1. By discarding the branch b_0, we obtain a simpler system

$$\begin{pmatrix} -I & \eta(n_1,n_2)I & \eta(n_1,n_3)I & \eta(n_1,n_4)I \\ \eta(n_2,n_1)I & \tilde{K}(n_2) - I & 0 & \eta(n_2,n_4)I \\ \eta(n_3,n_1)I & 0 & \tilde{K}(n_3) - I & \eta(n_3,n_4)I \\ \eta(n_4,n_1)I & \eta(n_4,n_2)I & \eta(n_4,n_3)I & -I \end{pmatrix} \begin{pmatrix} f(n_1) \\ f(n_2) \\ f(n_3) \\ f(n_4) \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ \eta(n_4,n_5) \end{pmatrix}.$$

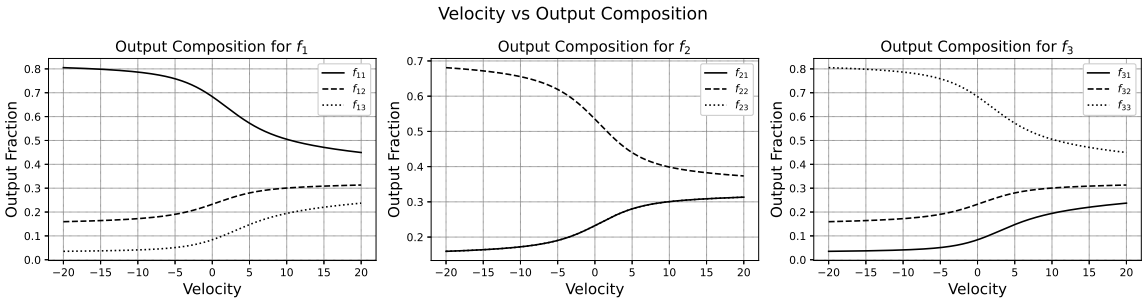


Figure 4.8.: Dependence of the $f_{ij}(n_0)$ on the advection velocity on b_5 for the network example of Figure 3.1. In the middle plot, the curves for f_{21} and f_{23} agree, so the latter is not apparent.

It makes sense, conceptually, and for the purpose of simplifying expressions, to introduce $h_i := \frac{1}{\tilde{\ell}(\mathbf{b}_i)}$, $h_{\bar{i}} = \frac{1}{\tilde{\ell}(\overline{\mathbf{b}}_i)}$. We may think of this reciprocal of the velocity-adjusted length as a measure of conductance: the larger h the easier it is to move along the corresponding $\mathbf{b}$.

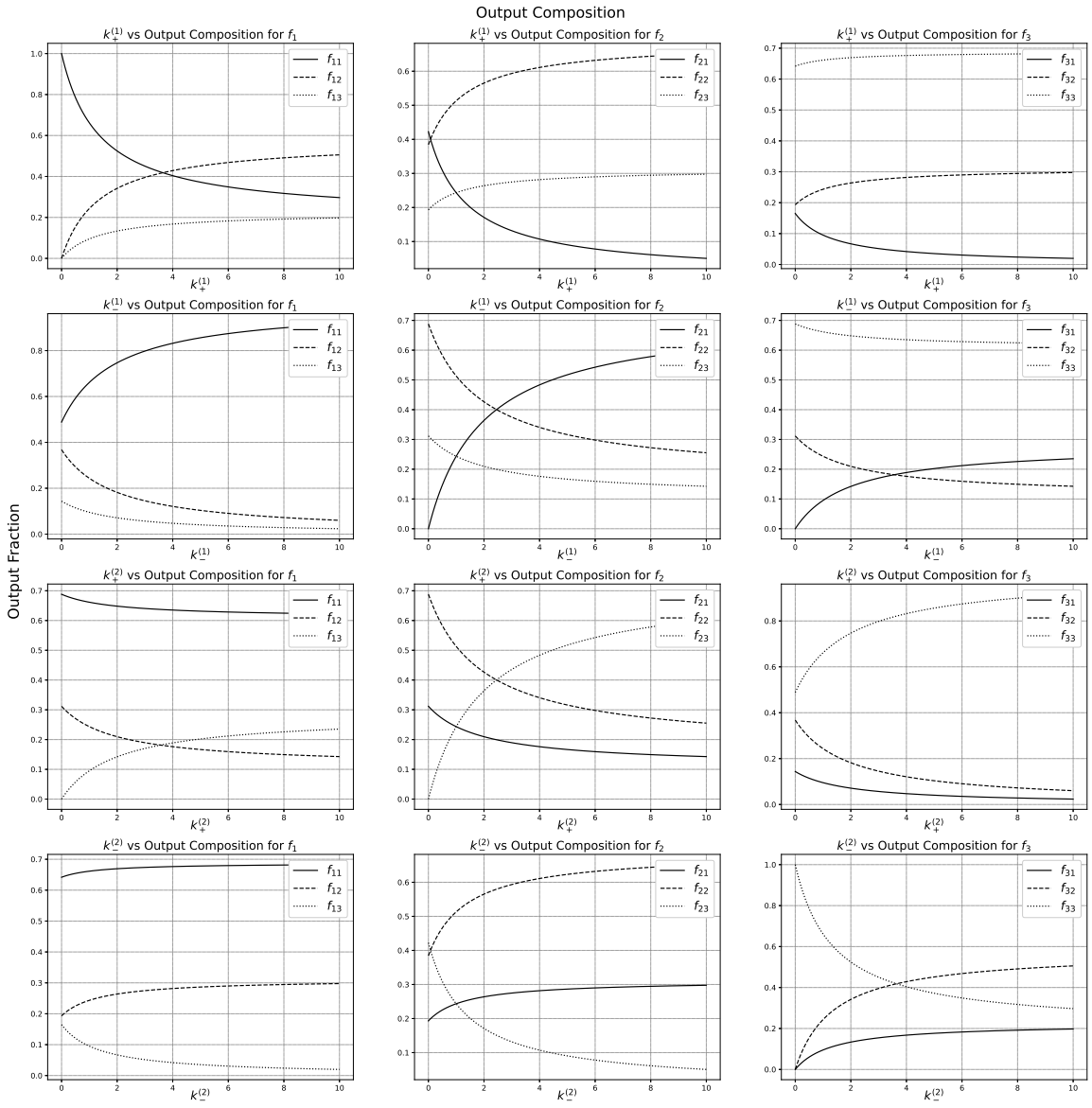


Figure 4.9.: Dependence of the f_{ij} on the reaction coefficients $k_\pm^1$ and $k_\pm^{(2)}$ for the network example of Figure 3.1.

Written in terms of conductances, the above system becomes

$$\begin{pmatrix} -I & \frac{h_1}{h_1+h_3+h_{\overline{5}}}I & \frac{h_3}{h_1+h_3+h_{\overline{5}}}I & \frac{h_{\overline{5}}}{h_1+h_3+h_{\overline{5}}}I \\ \frac{h_{\overline{1}}}{h_{\overline{1}}+h_2}I & \frac{2K(n_2)}{D(h_{\overline{1}}+h_2)} - I & 0 & \frac{h_2}{h_{\overline{1}}+h_2}I \\ \frac{h_{\overline{3}}}{h_{\overline{3}}+h_4}I & 0 & \frac{2K(n_3)}{D(h_{\overline{3}}+h_4)} - I & \frac{h_4}{h_{\overline{3}}+h_4}I \\ \frac{h_5}{h_{\overline{2}}+h_{\overline{4}}+h_5+h_6}I & \frac{h_{\overline{2}}}{h_{\overline{2}}+h_{\overline{4}}+h_5+h_6}I & \frac{h_{\overline{4}}}{h_{\overline{2}}+h_{\overline{4}}+h_5+h_6}I & -I \end{pmatrix} \begin{pmatrix} f(n_1) \\ f(n_2) \\ f(n_3) \\ f(n_4) \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{h_6}{h_{\overline{2}}+h_{\overline{4}}+h_5+h_6} \end{pmatrix}.$$

In order to keep the number of parameters reasonably small, let us suppose that all branch lengths $\ell(b) = \ell$ are equal and that the only non-zero advection velocity is on b_5. (See Figure 3.1.) Let us further choose the following reactions at nodes n_2 and n_3:

$$1 \underset{\mathrm{k}_-^{(1)}}{\overset{\mathrm{k}_+^{(1)}}{\rightleftharpoons}} 2 \text{ at } n_2, \quad 2 \underset{\mathrm{k}_-^{(2)}}{\overset{\mathrm{k}_+^{(2)}}{\rightleftharpoons}} 3 \text{ at } n_3.$$

We thus have 5 parameters (keeping the value of ℓ fixed), which can be written in dimensionless form:

$$s := \frac{\ell\nu(\mathbf{b}_5)}{D}, \quad \tilde{k}_\pm^{(1)} := \frac{\ell k_\pm^{(1)}}{D}, \quad \tilde{k}_\pm^{(2)} := \frac{\ell k_\pm^{(2)}}{D}.$$

The pairs $\tilde{k}_\pm^{(1)}$ and $\tilde{k}_\pm^{(2)}$ are a type of Damkoehler numbers.

The velocity-adjusted length of $\mathbf{b}_5$ is then a function of s:

$$\tilde{\ell}(\mathbf{b}_5) = \begin{cases} \frac{1-e^{-s}}{s}\ell & \text{if } s > 0 \\ \\ \frac{e^{|s|}-1}{|s|}\ell & \text{if } s < 0. \end{cases}$$

For very large $|\nu(\mathbf{b}_5)|$, if the velocity is positive, the adjusted length is very short and the nodes n_1 and n_2 effectively collapse into one; if the velocity is negative, the adjusted length is very long and this middle branch connecting n_1 and n_2 is effectively removed.

The plots in Figures 4.8 and 4.9 show how the fractions $f_{ij}(n_0)$ vary as functions of the above 5 parameters. We have fixed $\ell = 1$ and $D = 1$ so that $s = \nu(\mathbf{b}_5)$ and $\tilde{\kappa}_{\pm}^{(i)} = \kappa_{\pm}^{(i)}$. It is clear from the graphs that $\sum_j f_{ij}(n_0) = 1$.

5. The theory behind the solution to the OCP

The proof of our main result uses general facts from the theory of systems of linear parabolic partial differential equations. Justification for the claims made here, when not provided, can be derived from the standard literature on parabolic systems. The general results that we rely on are concerned with the existence and uniqueness of fundamental solutions, positivity of solutions, and exponential decay as time goes to ∞ under the assumption of zero Dirichlet condition on the exit boundary of the bounded domain $\mathcal{R}$. The main sources we have used are: [23, 26, 27]. In applications to Chemical Engineering, these general properties are typically taken for granted. For example, it is considered obvious that an open reactor in which substances undergo diffusion transport will eventually empty out, and concentrations will never be negative. For this reason, we do not elaborate on such technical issues and limit ourselves to providing the main line of argument to justify that, under the assumption that concentrations satisfy the given linear system of reaction-advection-diffusion equations, the output composition matrix-valued function satisfies the claimed boundary-value problem for an elliptic system.

One point of greater interest in applications that we do not explore here concerns estimates on the approximation of network-like reactors by network reactors. Although the argument we provide (see, in particular, Section 5.4.1 below) is qualitative, it should be possible to extract from it quantitative error estimates in terms of the parameter ϵ. (See Section 3.4 and Figure 3.2.) Related issues can be found in [28]. Similar estimates, but from a stochastic processes

perspective, are found in [29, 30]. These references are useful models for the estimates that should apply to our situation, although our case is expected to be less technical.

5.1 The reaction-transport equations and their fundamental solution

We consider the reaction-transport equations

$$\frac{\partial c_i}{\partial t} + \nabla \cdot \mathbf{j}_i = \sum_j c_j \kappa_{ji},$$

where $\mathbf{j}_i = c_i \mathbf{v}_i - D_i \nabla c_i$ is the flux vector field of species $i = 1, \ldots, N$, subject to the boundary conditions:

- $c_i(x,t) = 0$ for x on the exit boundary of $\mathcal{R}$;
- the normal component of the flux, $\mathbf{n}(x) \cdot \mathbf{j}_i(x,t)$, equals zero on the reflecting boundary.

Setting up the notation

$$(\mathcal{L}c)_i = \nabla \cdot (D_i \nabla c_i - c_i \mathbf{v}_i) + \sum_j c_j \kappa_{ji},$$

we have the system of partial differential equations $\frac{\partial c}{\partial t} = \mathcal{L}c$, where $c = (c_i)$ is regarded as a vector-valued function of x and t. For a function $\phi(x)$, not necessarily representing a concentration, it will occasionally be useful to write $\mathbf{j}_{i,\phi} := \phi \mathbf{v}_i - D_i \nabla \phi$, the *flux* associated to ϕ and i.

The *fundamental solution* to the reaction-transport system of equations is the function $\rho_{ij}(x,t|y,s)$ $(t > s)$ representing the concentration of i at position and time (x,t) given that a unit pulse of j is injected into $\mathcal{R}$ at position and time (y,s). This means that

$$\frac{\partial \rho_{ij}}{\partial t} = \nabla \cdot (D_i \nabla \rho_{ij} - \rho_{ij} \mathbf{v}_i) + \sum_k \rho_{kj} \kappa_{ki}$$

holds on $\mathcal{R}$ and

- $\rho_{ij}(x,t|y,s) = 0$ for x on the exit boundary $\partial_{\text{exit}}\mathcal{R}$;
- $\mathbf{n}(x)\cdot(D_i(x)\nabla_x\rho_{ij}(x,t|y,s) - \rho_{ij}(x,t|y,s)\mathbf{v}_i(x)) = 0$ for x on the reflecting boundary $\partial_{\text{reflect}}\mathcal{R}$;
- $\int_{\mathcal{R}} \rho(x,t|y,s)\phi(y)\,dy \to \phi(x)$ as t approaches s from above.

The integrand in the last item should be interpreted as matrix product:

$$\sum_j \int_{\mathcal{R}} \rho_{ij}(x,t|y,s)\phi_j(y)\,dy \to \phi_i(x).$$

This means that

$$\rho_{ij}(x,t|y,s) \to \delta(x-y)\delta_{ij}$$

where $\delta(x)$ is Dirac's delta supported at 0 and δ_{ij} is Kronecker's delta. When the coefficients of the reaction-transport equation are independent of t, we have $\rho(x,t|y,s) = \rho(x,t-s|y,0)$.

It can be shown [31] (by the uniqueness of the fundamental solution) that ρ satisfies the Chapman-Kolmogorov equation

$$\rho(x,t|y,s) = \int_{\mathcal{R}} \rho(x,t|\xi,\tau)\rho(\xi,\tau|y,s)\,d\xi$$

where the integrand involves matrix multiplication and $s < \tau < t$. This relation can be expressed in operator form as follows. Define for $t > 0$ and a row vector-valued function $\phi(y) = (\phi_i(y))$

$$(P_t\phi)(x) := \int_{\mathcal{R}} \phi(y)\rho(y,t|x,0)\,dy.$$

Note that $(\phi\rho)_i = \sum_j \phi_j\rho_{ji}$. Using the Chapman-Kolmogorov equation it is not difficult to obtain the semigroup property (see [32] for example)

$$P_{t_1+t_2} = P_{t_1} \circ P_{t_2}.$$

To the operator semigroup is associated its generator [32], which is the operator $\mathcal{A}$ defined by

$$\mathcal{A}\phi := \lim_{t\to 0} \frac{P_t\phi - \phi}{t}$$

on a domain consisting of the functions ϕ for which the limit exists.

5.2 Relation between $\mathcal{A}$ and $\mathcal{L}$

We wish to characterize $\mathcal{A}$ as the (Hilbert space) adjoint of $\mathcal{L}$. For this, consider the Hilbert space of square-integrable vector-valued (complex) functions on $\mathcal{R}$ with the inner product

$$\langle \varphi, \psi \rangle = \sum_j \int_{\mathcal{R}} \overline{\varphi_j(x)} \psi_j(x)\, dx.$$

The differential operator $\mathcal{L}$ is more precisely defined on the dense subspace of this Hilbert space consisting of continuous functions $\varphi = (\varphi_i)$ such that each φ_i is continuous and has square-integrable (weak) derivatives up to second order. In addition, functions in the domain of $\mathcal{L}$ are zero on the exit boundary of $\mathcal{R}$ and have 0 normal flux component, $\mathbf{n}(x) \cdot \mathbf{j}_{i,\varphi_i}(x) = 0$, at all x on the reflecting boundary.

A standard integration by parts computation using the divergence theorem gives:

$$\langle \varphi, \mathcal{L}\psi \rangle = \sum_j \int_{\mathcal{R}} \overline{[\nabla \cdot (D_j \nabla \varphi_i) + \mathbf{v}_j \cdot \nabla \varphi_j + \sum_k \kappa_{jk} \varphi_k]} \psi_j \, dx$$

$$\sum_j \left(\int_{\partial_{\text{exit}}\mathcal{R}} \overline{\varphi}_j D_j \mathbf{n} \cdot \nabla \psi_j \, dA + \int_{\partial_{\text{reflect}}\mathcal{R}} \psi_j D_j \mathbf{n} \cdot \nabla \overline{\varphi}_j \, dA \right).$$

For $\langle \varphi, \mathcal{L}\psi \rangle$ to be a bounded linear functional on the domain of $\mathcal{L}$ it is necessary and sufficient that φ_j be zero on the exit boundary and $\mathbf{n} \cdot \nabla \varphi_j$ be zero on the reflecting boundary. Therefore, the adjoint operator $\mathcal{L}^*$ of $\mathcal{L}$ is the differential operator whose domain consists of continuous functions $\varphi = (\varphi_j)$ on $\mathcal{R}$, whose (weak) derivatives up to second order are square integrable,

having value 0 on the exit boundary and normal derivative 0 on the reflecting boundary. On these functions,

$$(\mathcal{L}^*\varphi)_j = \nabla \cdot (D_j \nabla \varphi_i) + \mathbf{v}_j \cdot \nabla \varphi_j + \sum_k \kappa_{jk} \varphi_k.$$

Notation: When applying $\mathcal{A}$ or $\mathcal{L}$ to $\rho(y, t|x, 0)$ in the below calculations, we use a subscript such as in $\mathcal{L}_x$ or $\mathcal{A}_y$ to indicate which variable it is being acted on.

We claim that $\mathcal{A} = \mathcal{L}^*$. This is seen as follows:

$$\begin{aligned}
(\mathcal{A}\varphi)(x) &= \lim_{t\to 0} \frac{1}{t} \left(\int_{\mathcal{R}} \varphi(y)\rho(y, t|x, 0)\, dy - \varphi(x) \right) \\
&= \lim_{\epsilon\to 0} \lim_{t\to 0} \int_{\mathcal{R}} \varphi(y) \frac{\rho(y, t+\epsilon|x, 0) - \rho(y, \epsilon|x, 0)}{t}\, dy \\
&= \lim_{\epsilon\to 0} \int_{\mathcal{R}} \varphi(y) \frac{\partial \rho}{\partial t}(y, \epsilon|x, 0)\, dy \\
&= \lim_{\epsilon\to 0} \int_{\mathcal{R}} \varphi(y) (\mathcal{L}_y \rho)(y, \epsilon|x, 0)\, dy \\
&= \lim_{\epsilon\to 0} \int_{\mathcal{R}} (\mathcal{L}^*\varphi)(y) \rho(y, \epsilon|x, 0)\, dy \\
&= \left(\mathcal{L}^*\varphi\right)(x).
\end{aligned}$$

A similar argument to the above integration by parts computation also shows that $\mathcal{A}^* = \mathcal{L}$. Since $\mathcal{A}$ is the relevant operator for the output composition problem, from this point on we dispense with the $\mathcal{L}, \mathcal{L}^*$ notation and only use $\mathcal{A}, \mathcal{A}^*$. We summarize below the definitions of $\mathcal{A}$ and

$\mathcal{A}^*$. They are both defined on continuous vector-valued functions on $\mathcal{R}$ whose weak second derivatives are square integrable and satisfy:

$$(\mathcal{A}\varphi)_j = \nabla \cdot (D_j \nabla \varphi_j) + \mathbf{v}_j \cdot \nabla \varphi_j + \sum_k \kappa_{jk} \varphi_k$$

$$\varphi_j = 0 \text{ on } \partial_{\text{exit}} \mathcal{R}$$

$$\mathbf{n} \cdot \nabla \varphi_j = 0 \text{ on } \partial_{\text{reflect}} \mathcal{R}$$

and

$$(\mathcal{A}^* \psi)_j = \nabla \cdot (D_j \nabla \psi_j - \psi_j \mathbf{v}_j) + \sum_k \psi_k \kappa_{kj}$$

$$\psi_j = 0 \text{ on } \partial_{\text{exit}} \mathcal{R}$$

$$\mathbf{n} \cdot (\psi_j \mathbf{v}_j - D_j \nabla \psi_j) = 0 \text{ on } \partial_{\text{reflect}} \mathcal{R}.$$

As already noted, $\mathcal{A}$ generates a one-parameter semigroup P_t such that

$$(P_t \varphi)(x) = \int_{\mathcal{R}} \varphi(y) \rho(y, t|x, 0)\, dy.$$

The adjoint semigroup is P_t^* is characterized by $\langle P_t^* \psi, \varphi \rangle = \langle \psi, P_t \varphi \rangle$. It is also an integral operator with (matrix) kernel denoted $\rho_{ij}^*(x, t|y, s)$. We summarize here the defining properties of these two integral kernels:

- $\rho(x, t|y, s)$, $t > s$, satisfies

$$\frac{\partial \rho_{ij}}{\partial t} = \nabla \cdot (D_i \nabla \rho_{ij} - \rho_{ij} \mathbf{v}_i) + \sum_k \rho_{kj} \kappa_{ki} = (\mathcal{A}^* \rho_{\cdot j})_i$$

where the derivatives are in x. Further, the boundary conditions $\rho_{ij}(x,t|y,s) = 0$ for x on the exit boundary and $\mathbf{n}\cdot(D_i\nabla\rho_{ij} - \rho_{ij}\mathbf{v}_i) = 0$ for x on the reflecting boundary and the initial condition

$$\int_{\mathcal{R}} \rho(x,t|y,s)\varphi(y)\,dy \to \varphi(x) \text{ as } t \downarrow s$$

hold for any given φ.

- $\rho^*(x,t|y,s)$, $t < s$, satisfies

$$-\frac{\partial \rho^*_{ij}}{\partial t} = \nabla\cdot(D_i\nabla\rho^*_{ij}) + \mathbf{v}_i\cdot\nabla\rho^*_{ij} + \sum_k \kappa_{ik}\rho^*_{kj} = (\mathcal{A}\rho^*_{\cdot j})_i$$

where the derivatives are in x. Further, the boundary conditions $\rho^*_{ij}(x,t|y,s) = 0$ for x on the exit boundary and $\mathbf{n}\cdot\nabla\rho^*_{ij} = 0$ for x on the reflecting boundary and the initial condition

$$\int_{\mathcal{R}} \rho^*(x,t|y,s)\varphi(y)\,dy \to \varphi(x) \text{ as } t \uparrow s$$

hold for any given φ.

It can be further verified using a relatively standard argument for systems of parabolic partial differential equations (see [23]) that

$$\rho^*(y,s|x,t) = \rho(x,t|y,s)^\top$$

where $\top$ indicates matrix transpose. Finally, as the coefficients $D_i, \mathbf{v}_i, \kappa_{ij}$ do not depend on time t explicitly, we have

$$\rho(x,t|y,s) = \rho(x,t-s|y,0).$$

5.3 The output composition matrix

The main quantity of interest in this work is the total amount of species i which is produced after the reactor is fully evacuated. Let us represent this quantity by A_i. In analytic form,

$$A_i = \lim_{T\to\infty} \int_0^T \int_{\partial_{\text{exit}}\mathcal{R}} \mathbf{n}(x) \cdot \mathbf{j}_i(x,t)\, dA(x)\, dt.$$

Here $\partial_{\text{exit}}\mathcal{R}$ indicates the exit boundary of the reactor and $dA(x)$ is the element of surface area. (The volume element will be written simply dx.) The integral over the exit boundary gives the rate of flow of species i out of the reactor and its integral over the time interval $[0,T]$ is the amount of i that escapes by time T. An application of the divergence theorem (using that $\mathbf{n}(x)\cdot\mathbf{j}_i(x,t) = 0$ on the reflecting boundary of $\mathcal{R}$) together with the reaction-transport equation yields

$$A_i = \lim_{T\to\infty} \int_0^T \int_{\mathcal{R}} -\frac{\partial c_i}{\partial t}(x,t)\, dx\, dt + \sum_j \int_0^\infty \int_{\mathcal{R}} c_j(x,t)\kappa_{ji}(x)\, dx\, dt.$$

Note that

$$\int_0^T \int_{\mathcal{R}} \frac{\partial c_i}{\partial t}(x,t)\, dx\, dt = \int_0^T \frac{d}{dt} \int_{\mathcal{R}} c_i(x,t)\, dx\, dt = Q_i(T) - Q_i(0),$$

where $Q_i(t) := \int_{\mathcal{R}} c_i(x,t)\, dx$ is the quantity of i still left in the reactor by time t. Since in the long run the reactor is fully evacuated,

$$A_i = Q_i(0) + \sum_j \int_0^\infty \int_{\mathcal{R}} c_j(x,t)\kappa_{ji}(x)\, dx\, dt. \tag{5.1}$$

Let $A_{ij}(y)$ indicate the amount (molar fraction) of species i produced by the system given that, at the initial time, a unit pulse of j is injected into the reactor at position y. This quantity can be expressed using the fundamental solution as

$$A_{ij}(y) = \delta_{ij} + \sum_k \int_0^\infty \left[\int_{\mathcal{R}} \rho_{kj}(x,t|y,0)\kappa_{ki}(x)\, dx \right] dt.$$

In matrix form, and slightly simplifying the notation $\rho(x,t|y) \coloneqq \rho(x,t|y,0)$,

$$\begin{aligned} A(y) &= I + \int_0^\infty \int_{\mathcal{R}} \kappa^\top(x)\rho(x,t|y)\,dx\,dt \\ &= I + \int_0^\infty \int_{\mathcal{R}} \kappa^\top(x)\rho^*(y,-t|x)^\top\,dx\,dt. \end{aligned}$$

Here I is the identity $N \times N$-matrix. The transpose of $A(y)$ is

$$A^\top(y) = I + \int_0^\infty \int_{\mathcal{R}} \rho^*(y,-t|x)\kappa(x)\,dx\,dt.$$

Note that $\mathcal{A}I = \kappa(y)$ and that $\frac{\partial}{\partial t}\rho^*(y,-t|x) = \mathcal{A}\rho^*(y,-t|x)$.

We can now state the boundary-value problem satisfied by $A^\top(y)$.

$$\begin{aligned} (\mathcal{A}A^\top)(y) &= \kappa(y) + \int_0^\infty \int_{\mathcal{R}} \mathcal{A}_y\rho^*(y,-t|x)\,\kappa(x)\,dx\,dt \\ &= \kappa(y) + \int_0^\infty \frac{d}{dt}\int_{\mathcal{R}} \rho^*(y,-t|x)\kappa(x)\,dx\,dt \\ &= \kappa(y) + \lim_{\epsilon\to 0}\lim_{T\to\infty}\int_{\mathcal{R}} \left[\rho^*(y,-T|x) - \rho^*(y,-\epsilon|x)\right]\kappa(x)\,dx. \end{aligned}$$

Now observe that $\rho^*(y,-T|x) = \rho(x,T|y,0)^\top$ will tend towards 0 as $T \to \infty$. This is because, in the long run, the open reactor will be fully emptied. Furthermore $\rho^*(y,-\epsilon|x) \to \delta(x-y)I$ as $\epsilon \to 0$. We conclude that $(\mathcal{A}A^\top)(y) = \kappa(y) - \kappa(y) = 0$.

When y approaches the exit boundary, $\rho(x,t|y)$ approaches 0, hence $A_{ij}(y) \to \delta_{ij}$. It also follows from the general properties of ρ and ρ^* that $A^\top$ satisfies the Neumann condition on the reflecting boundary. Thus we obtain the following central result. (We now use x for the position variable in A.)

Conclusion (Boundary-value problem for the output composition matrix)**.** *The transpose $f(x) = A^\top(x)$ of the matrix-valued output composition function $A(x)$ satisfies the equation $\mathcal{A}f = 0$ on*

$\mathcal{R}$ together with the Neumann condition on the reflecting boundary of $\mathcal{R}$ and $f(x) = I$ on exit boundary points.

5.4 Reduction to network reactors

We wish now to rewrite the boundary-value problem for the solution to the OCP so that it applies to network reactors. The main assumptions are that the transport coefficients D_i and $\mathbf{v}_i$ are constant on cross-sections of reactor pipes (see Figure 3.2) and that the output composition matrix $f(\mathbf{x}) = A^{\intercal}(\mathbf{x})$ is well-approximated by functions that are constant on those cross-sections. Due to the Neumann condition on the reflecting boundary, this assumption can be expected to hold well if pipes are long and narrow. Recalling that the reaction coefficients are zero outside junctures, then the partial differential operator $\mathcal{A}$ reduces to

$$\mathcal{A}\varphi_i = \nabla \cdot (D_i \nabla \varphi_i) + \mathbf{v}_i \cdot \nabla \varphi_i = \frac{d}{dx}\left(D_i^b \frac{d\varphi_i}{dx}\right) + \nu_i^b \frac{d\varphi_i}{dx} = 0,$$

where x is arc-length parameter along the branch b (the center axis of the pipe) and a choice of orientation of b has been made so that the sign of ν_i^b is defined. To these equations (indexed by i and b) we need to add conditions on nodes, obtained as junctures are imagined to shrink to points. It is natural to suppose that, in the limit, f is still continuous at nodes (its value at a node coincides with the values of the limits as the node is approached from the direction of any of the branches attached to it.) We further need to specify conditions on the derivatives of f along branch directions at a node. This is done next.

5.4.1 Node conditions

The conditions on f and its first directional derivatives at a node n can be determined by integrating the (matrix) equation $\mathcal{A}f = 0$ over the juncture indexed by n, applying the divergence theorem, and recalling the relationship between κ and K described in Subsection 3.4. (See the notation described in Figure 3.2.) Let us write $\varphi_i = f_{ij}$ for a fixed j. Thus we integrate term-by-term the equation

$$\nabla \cdot (D_i \nabla \varphi_i) + \mathbf{v}_i \cdot \nabla \varphi_i + \sum_k \kappa_{ik} \varphi_k = 0$$

on the juncture $\mathcal{R}_\epsilon(n)$. Since φ_i has zero normal derivative at the reflecting boundary of $\mathcal{R}$,

$$\int_{\mathcal{R}_\epsilon(n)} \nabla \cdot (D_i \nabla \varphi_i)\, dV = \sum_b \int_{\mathcal{A}_\epsilon(n,b)} D_i \nabla \varphi_i \cdot \mathbf{u}_b(n)\, dA \approx \sum_b A_\epsilon(n,b) D_i^b(n) \frac{d\varphi_i^b}{dx}(n).$$

Here $\mathcal{A}_\epsilon(n,b)$ is the disc component of the boundary of $\mathcal{R}_\epsilon(n)$ that attaches to the pipe indicated by b and, as already defined, $A_\epsilon(n,b)$ is the area of this disc. We assume that the parametrization is such that $x = 0$ corresponds to n. V and A are the volume and area elements in integration.

The volume integral of $\mathbf{v}_i \cdot \nabla \varphi_i$ is proportional to the volume $V_\epsilon(n)$ of the juncture. Let us write it as $C_\epsilon(n) V_\epsilon(n)$, where $C_\epsilon(n)$ is the average value of the integrand on $\mathcal{R}_\epsilon(n)$. The integral of $\kappa_k \varphi_{ik}$ can be approximated using the characterization of $K_{ik}(n)$ (see Section 3.4):

$$\int_{\mathcal{R}_\epsilon(n)} \kappa_{ik}(\mathbf{x}) \varphi_k(\mathbf{x})\, dV \approx A_\epsilon(n) K_{ik}(n) \varphi_k(n).$$

Here we recall that $A_\epsilon(n)$ is the sum of the $A_\epsilon(n,b)$ over the branches b attached to n. Adding these three terms and dividing by $A_\epsilon(n)$ (recall that $p(n,b) = A_\epsilon(n,b)/A_\epsilon(n)$), we obtain, to first order in ϵ,

$$\sum_b p(n,b) D_i^b(n) \frac{d\varphi_i^b}{dx}(n) + C_\epsilon(n) \frac{V_\epsilon(n)}{A_\epsilon(n)} + \sum_k K_{ik}(n) \varphi_k(n) = 0.$$

Now recall that $\epsilon A_\epsilon(n)/V_\epsilon(n)$ converges to a dimensionless number $\chi(n)$ characteristic of the juncture geometry. Thus $V_\epsilon(n)/A_\epsilon(n)$ is of the order ϵ. Eliminating this term and passing to the limit as ϵ approaches 0 yields

$$\sum_b p(n,b) D_i^b(n) \frac{d\varphi_i^b}{dx}(n) + \sum_k K_{ik}(n)\varphi_k(n) = 0. \tag{5.2}$$

5.4.2 Invertibility of the matrix Λ

Recall the definition of the matrix Λ in Equations (3.15) or (3.17). We wish to provide here sufficient conditions for Λ to be invertible based on the form of this matrix, independently of the boundary-value problem from which it originated. Let λ_{ij} be the (i,j)-matrix element of Λ. We first note the following two properties of these elements:

1. For each row of Λ with row index i we have $|\lambda_{ii}| \geq \sum_{j:j\neq i} |\lambda_{ij}|$.

2. For at least one i (in fact, for those i for which $\eta(n_i, n_{\text{exit}}) > 0$), $|\lambda_{ii}| > \sum_{j:j\neq i} |\lambda_{ij}|$.

In checking property 1, notice that $\sum_j \eta(n_i, n_j) = 1$ and $\sum_s \tilde{K}_{rs}(n_i) = 0$.

We now make the assumption that the network reactor is connected in the following sense: if we remove the exit nodes as well as all the branches attached to those nodes, then the resulting network does not disconnect. There is no loss of generality in making this assumption since, otherwise, the output composition matrix would only depend on the connected piece that received the initial injection of reactants.

Define $\tilde{K} = \sum_n \tilde{K}(n)$, where the sum is over the active nodes of the network reactor. Invertibility of Λ can be established by general facts about matrices if we make the further assumption that any substance among the N species can be transformed into any other by

a sequence of reactions among those encoded in $\tilde{K}$. This assumption on $\tilde{K}$ together with connectivity of the network imply that Λ is an irreducible matrix in the sense of definition 6.2.25 of [33]. Irreducibility plus the above properties 1 and 2 are the conditions needed to apply Corollary 6.2.27 in [33], which implies the desired invertibility result.

It was tempting to try to use the Gershgorin circle theorem to prove the invertibility, by showing that the eigenvalue 0 is not in any Gershgorin disc. The problem with using the Gershgorin circle theorem directly is that the eigenvalue 0 may be on the boundary of the Gershgorin discs, even though it cannot be on the interior. Corollary 6.2.27 gives the stronger result that the eigenvalue 0 cannot be on the boundary, and so the matrix is invertible.

6. Re-Tuning: Overcoming the Compositionality Limits of Large Language Models with Recursive Tuning

6.1 Background

Large language models such as ChatGPT [34] have recently seen numerous successes [35]. These include passing the bar exam, scoring high on AP exams, translating various languages and more. Many people using ChatGPT and other models can attest to the productivity improvements these models bring.

However, there are still many tasks that language models cannot perform despite their successes. [36] evaluate the performance of language models, including GPT4, on 3 compositional tasks. These are a puzzle problem, a dynamic programming problem, and multiplication. All of these problems have relatively straightforward algorithms the language model can follow to solve the problem. Even when giving the model a prompt that describes the procedure to solve these problems, language models fail for large input sizes.

Indeed, [36] find that for small input sizes, for example multiplying 2-digit numbers, language models are able to successfully follow the correct steps to solve the problem. For larger problem sizes, like multiplying 5-digit numbers, the language models have little chance at following

all the steps to solve the problem correctly. GPT4 has 99% accuracy at multiplying 2 2-digit numbers but 0% accuracy at multiplying 2 5-digit numbers.

6.1.1 Length Generalization

Additional studies have noted the difficulties of language models on 3 tasks we will consider: addition [37, 38], parity [39], and the dynamic programming problem of [36]. In particular, language models can be trained to do well on in-distribution examples of these problems, but fail to generalize to out-of-distribution (OOD) examples. Here we define in-distribution and out-of-distribution by the lengths of problems. Our training setup is to train a model on problems up to a certain length and then evaluate the model on problems with greater lengths. The "in-distribution" problems are the problems with length in the training set and the "out-of-distribution" problems are those that are longer than the problems in the training set. For example, a model may be trained on addition problems up to 15 digits. "In-distribution" problems for this model would be addition problems where each number is between 1-15 digits and OOD problems would be those that have numbers with 16 or more digits. Our goal is to train models that do well on OOD problems.

On addition, [37], [38] and others look at length generalization. In [37], they train their model to perfect accuracy on addition up to 8 digits. Then the evaluate performance on 9 and 10-digit addition. By 10 digits, their accuracy falls to 50%. In [38], they train their model to perfect accuracy on addition up to 15 digits, and evaluate on addition up to 21 digits. By 21 digits their accuracy falls to 0%. In contrast, our Re-Tuning model is also trained on addition

up to 15 digits, but maintains near perfect accuracy at 21 digits and still has 50% accuracy at adding 60-digit numbers.

6.2 Introduction

We present a new method for large language models to solve compositional tasks. Although they have shown strong performance on traditional language understanding tasks, large language models struggle to solve compositional tasks, where the solution depends on solutions to smaller instances of the same problem. We propose a natural approach to solve the task recursively. Our method, Re-Tuning, tunes models to divide a problem into subproblems, solve those subproblems, and combine the results. We show that our method significantly improves the model performance on three representative compositional tasks: integer addition, dynamic programming, and parity. Compared to state-of-the-art methods that keep intermediate steps towards solving the problems, Re-Tuning achieves significantly higher accuracy and is more GPU memory efficient.

Large language models (LLM) have obtained the state-of-the-art performance on a wide set of tasks [34, 40–45]. However, recent studies [36, 39, 46] show these models struggle to generalize to compositional tasks, where the solution depends on solutions to smaller instances of the same problem. An example task, integer addition, is shown in Figure 6.1. When calculating "1,234+4,567", we first break the problem into a smaller problem "234+567". After obtaining the answer to this problem, the original problem is partially solved. Similarly, to get the result of "234+567", we further divide it into solving "34+67". This recursion is the fundamental

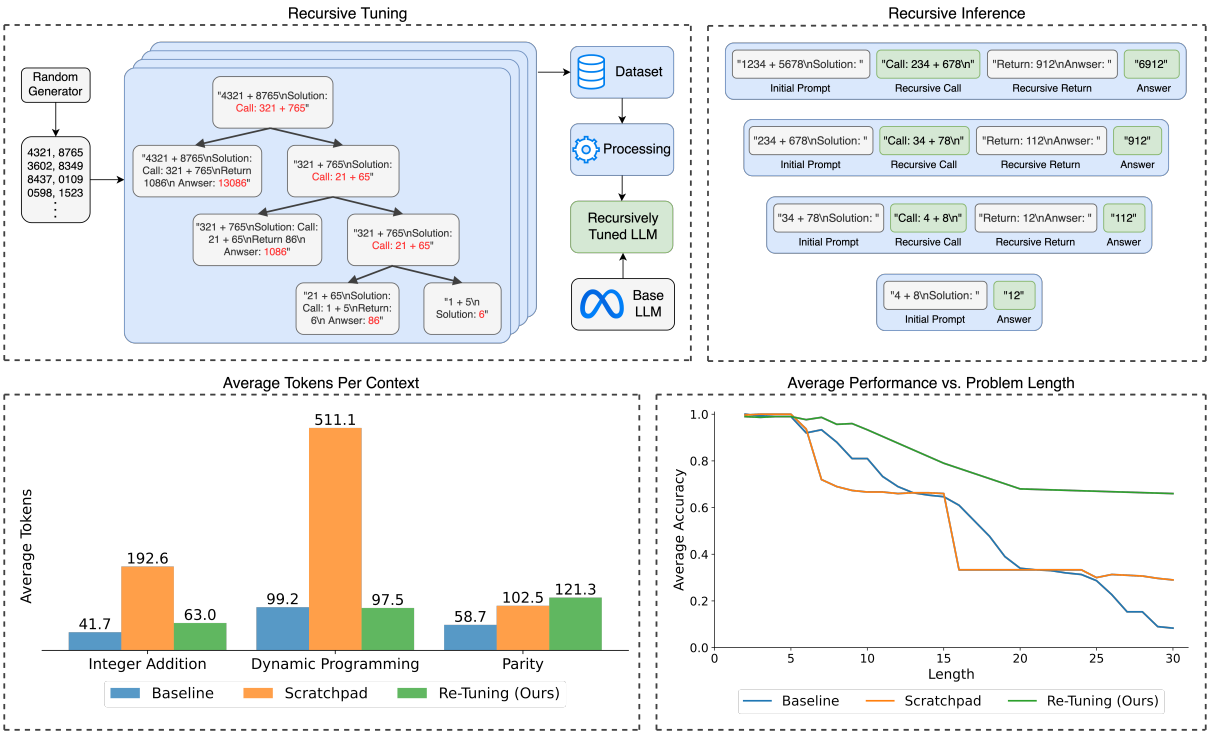


Figure 6.1.: Summary of our approach and results. Upper Left: Our recursive tuning pipeline generates and processes all the recursive subproblems for each randomly generated problem instance in order to train the base LLM. Upper Right: Our recursive inference pipeline allows the model to call itself on a subproblem of reduced size, which enables the subproblem to be solved in a new context, and return the answer back to the initial context. Lower Left: On most problems, Re-Tuning trains on significantly less tokens than the scratchpad method, saving considerable GPU memory. Bottom Right: On average, Re-Tuning outperforms the baseline and scratchpad methods across all tasks, especially as the problems grow in size and complexity.

operation to solve compositional tasks. However, none of the existing approaches have explicitly captured the recursive nature of compositional tasks yet.

In this part of the thesis, we propose a recursion-based method for LLMs to improve their ability to solve compositional tasks. Intuitively, we tune LLMs to divide problems into subproblems of the same type as the original problem, solve those subproblems, and combine the results (Figure 6.1). More specifically, we adopt a top-down approach to solve the original problems recursively. During training, we finetune LLMs on smaller and smaller instances of the original problems, during which LLMs learn to call themselves recursively on a smaller version of the problem until reaching the base case. We empirically set the base case for different tasks. LLMs are taught to devise the solution to the problem by combining the solutions obtained from the simpler versions of the problem. We then deploy the recursively tuned LLMs to solve problems at inference time. The above procedure is referred as recursive tuning (or Re-Tuning in short).

The basic idea behind Re-Tuning is motivated by two lines of work. First, recent work [36, 37, 39] show that training LLMs on high-quality scratchpad data, which includes intermediate steps towards solving a problem, can improve performance on certain compositional tasks such as integer addition and parity. Instead of using the intermediate steps to finetune models, which adds additional computation cost to the original problem, Re-Tuning divides the problems into smaller and smaller instances. Each of the instances runs independently with its own context in the associated call stack. The problem size in each context is actually reduced. The solution to each subproblem is then propagated up in the call stack to produce the final solution. A side product is that this allows models to more easily attend to the context, improving the accuracy of solving each subproblem. Second, our tuning process is reminiscent of recent works that incorporate tool use in LLMs [47, 48]. Similar to how these models call a tool and resume

generating final output based on the output of the tool, in Re-Tuning, the models call themselves on a subproblem and resume generating after receiving the solution to the subproblem.

We empirically evaluate the performance of Re-Tuning on three representative compositional tasks: integer addition [46], a dynamic programming problem [36], and the parity problem [39, 46]. Our results show Re-Tuning improves the average performance of LLaMA 7B and LLaMA 13B on all tasks by 37.4% and 31.9% over baseline finetuning. Compared to scratchpad finetuning, our improvement is striking, with averaging 34.5% and 36.7% points improvements on LLaMA 7B and LLaMA 13B respectively. Importantly, we show Re-Tuning saves significant GPU memory compared to the scratchpad method when training. We hope our results foster future research on recursive learning of large foundation models.

6.3 Approach

We present Re-Tuning in this section. Re-Tuning recursively tunes LLMs to solve compositional tasks. Specifically, the method (1) recursively decreases the size of the problem, (2) solves the base case, and (3) passes the solutions up the recursion stack, solving subproblems of increasing complexity along the way.

First, Re-Tuning recursively generates prompts for subproblems of decreasing length or complexity. For example, when adding the two numbers 1,234 and 5,678, the model generates a prompt to add 234 + 678. Then this prompt is sent to a separate context where the model generates a new prompt to add 34 + 78, and finally this new prompt is sent to a separate context where the model again generates a new prompt to add 4 + 8.

Next, the base case is solved. Certainly, the base cases are trivial enough to be solved directly in the same context. For the integer addition problem, the base case is to add the two least-significant digits together, for example, 4 + 8.

Finally, the solutions are passed up a level in the recursive stack. They are appended after the prompt in the previous context in the recursive stack. Again, sticking with integer addition, it's helps to know the solution to 4 + 8 when tasked with solving 34 + 78. So the answer the model generates to 4 + 8 is appended to the context tasked with solving 34 + 78. This process of appending solutions continues until eventually, the solution to the first recursive call is passed to the initial context, which helps to solve the initial prompt.

In order to accomplish this, we finetune the model to (1) generate recursive subproblems, (2) solve base cases, and (3) to use the answers propagated up from these recursive calls in its computation. During generation, the model can designate some of its generated text to be a prompt by enclosing the text between '`Call: `' and '`\n`'. Once this happens, we stop generating in the current context and prompt the model with the enclosed text in a new context. In each new context we follow the exact same generation procedure, except for the base case where the model learns to directly output the answer to the base case rather than making another recursive call. When generation in the new context is complete, we take the final answer, which is separated by the text '`\nAnswer: `', and append that to the initial context which generated the prompt. Then we continue generation in the initial context. Basic pseudo-code for the generation procedure is in Figure 9.4.

In the addition example, the model only needs to generate one recursive call in each context. However, our method works more generally than this. Many recursive calls may be generated in

a given context. For example, the dynamic programming problem we describe below requires multiple recursive calls in the initial context to solve the problem.

7. Re-Tuning Experimental Results

7.1 Experiments

We consider three tasks: integer addition, a dynamic programming problem, and the parity problem. For each task, we train 3 types of models: baseline, scratchpad, and Re-Tuning. The baseline models were trained to simply output the solution to the problem. The scratchpad models were trained to generate a scratchpad [37] containing intermediate reasoning steps before generating the final solution to the problem. The Re-Tuning models are as described above.

In our evaluation, we look at in-distribution and out-of-distribution (OOD) data. The in-distribution data are those with lengths that were seen in training and the OOD data are those with lengths longer than seen in training. For example, on addition the training data consists of numbers with lengths up to 15 digits. So evaluation examples with 1-15 digits are considered in-distribution and examples with 16 or more digits are considered OOD.

We finetune LLaMA 7B and 13B [44] using Low-Rank Adapters [49].

7.1.1 Experimental Setup

Tasks and Datasets

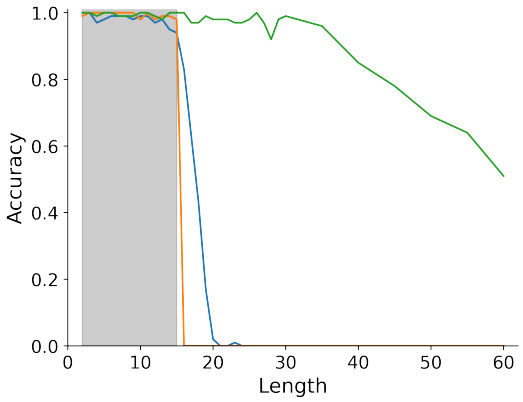
We consider 3 representative compositional problems: integer addition, a dynamic programming problem, and the parity problem. Here, we describe each problem in detail, as well as how the data was constructed.

Integer Addition With this problem, we investigate the extent to which LLMs can add two integers. The input to the model is simply the prompt to add 2 numbers. For example, "45 + 97". Language models have some capability to perform addition without any finetuning, but it seemingly disappears as the numbers grow in size. [37] used a scratchpad to teach language models addition, and more recently [38] taught LLaMA 7B to add numbers up to 15 digits. In both cases, there is remarkable performance degradation when adding integers larger than those seen during finetuning. [46] suggest that addition is particularly hard for LLMs since it requires precise indexing operations and the non-causal propagation of the carry term.

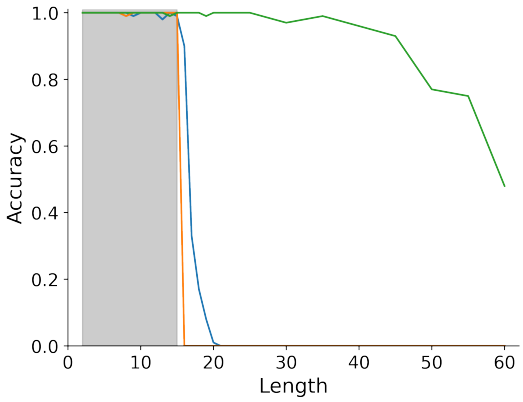
Following [38], we generate training data summing randomly-generated integers up to 15 digits long. During evaluation, we sample 100 problems of lengths up to 60 digits.

Dynamic Programming Problem We leverage the dynamic programming problem recently studied by [36]:

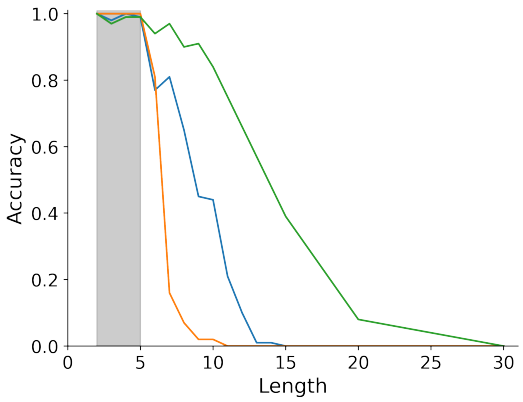
> "Given a sequence of integers, find a subsequence with the highest sum, such that no two numbers in the subsequence are adjacent in the original sequence."



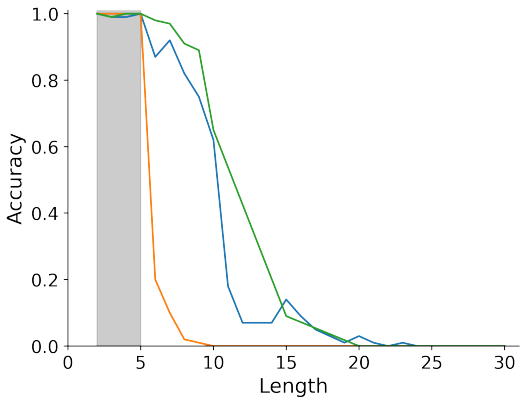
(a) LLaMA 7B Integer Addition



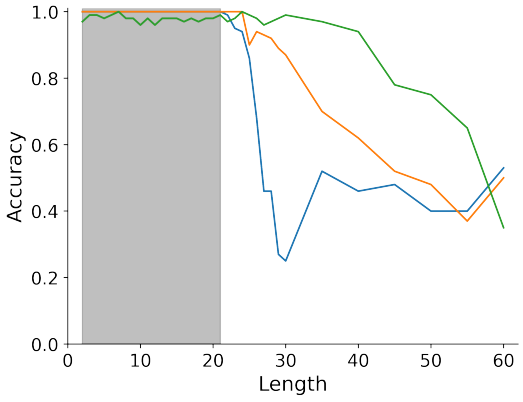
(b) LLaMA 13B Integer Addition



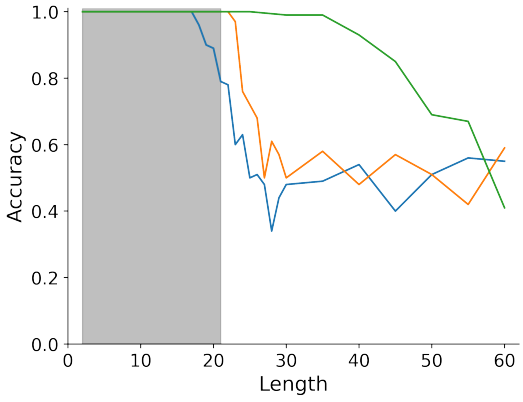
(c) LLaMA 7B Dynamic Programming



(d) LLaMA 13B Dynamic Programming



(e) LLaMA 7B Parity



(f) LLaMA 13B Parity

Figure 7.1.: Performance of LLaMA 7B (left) and LLaMA 13B (right) on Addition, Dynamic Programming, and Parity. The in-distribution range is shaded in gray.

This problem can be broken down into two steps: first, recursively generate an array of sub-array sums, and then recursively identify which indices correspond to the highest sum. Re-Tuning generates prompts for each of these steps, which are then solved in recursive contexts.

For example, consider the sequence $[3, 2, -2, 5, 3]$. The subsequence with the highest sum with no adjacent numbers would be the one that has the first 3 and the 5. For this, the model outputs a list of 1s and 2s with 1s corresponding to numbers chosen and 2s corresponding to numbers not chosen. In the case, the model should output $[1, 2, 2, 1, 2]$.

Following [36], we exhaustively generate all permutations of arrays up to length 5 for training, where each element is restricted to $[-5, 5]$. Evaluation is done on arrays up to length 30, still with each integer element restricted to $[-5, 5]$.

Parity Problem The parity problem is to determine if there is an even or odd number of 1's in an input array consisting of 0's and 1's. An example input is $[0, 1, 0, 0]$, for which the output should be 1 since the array contains an odd number of 1's. In cases where there is an even number of 1's the output should be 0. This problem was previously studied by [39] and [46], who explored length generalization on this problem. This problem can be solved by traversing the input array and adding all the numbers mod 2, which is the method we finetune our model to use.

We generate binary arrays up to length 21 for training. For evaluation, we sample 100 arrays from various lengths up to 80.

7.1.2 Main Results

Here we share our main results on LLaMA 7B and LLaMA 13B, across all three tasks. Results are shown in Figure 7.1, and discussed in detail in the proceeding sections. Across all problems and model sizes, the Re-Tuning method outperforms the baseline and scratchpad methods, with the clearest difference being on the addition. The baseline method tends to exhibit better OOD generalization compared to scratchpad among the different problems.

Integer Addition

In Figure 7.1, we can see the Re-Tuning method outperforms baseline and scratchpad methods. The scratchpad method performs the worst, achieving 0% accuracy on every problem longer than those seen during training. The baseline method has non-zero OOD accuracy for problems upto length 20, but still quickly falls to 0% accuracy on longer problems. In contrast, the Re-Tuning method maintains near-perfect accuracy in regimes where the baseline and scratchpad models have 0% accuracy, and maintains around 50% accuracy on adding up to 60 digit numbers. The model from [38], which is also trained on addition up to 15 digits, has similar OOD performance to our baseline model, and falls to 0% accuracy when adding 21 digit numbers.

Dynamic Programming Problem

Again, the Re-Tuning outperforms both baseline and scratchpad approaches, though the gap between Re-Tuning and baseline is narrower for LLaMA 13B than it is for LLaMA 7B. [36] also evaluate a finetuned GPT3 with and without scratchpad, which are also trained on examples

up to length 5. Both of their models reach 0% accuracy when evaluating on length 10, which is similar to our baseline and scratchpad results.

Parity Problem

Re-Tuning performs as well or better on inputs up to (but not including) size 60. Since the baseline and scratchpad methods only every learn to output either 0 or 1, it can maintain an accuracy of around 50%, or random chance. The Re-Tuning models learn to output text other than just 0 or 1, such as recursive calls, and so they can fall below 50% accuracy on OOD data. Up to length 60, Re-Tuning is still the dominant performer on the problem.

7.2 Analysis and Further Discussion

In this section, we conduct additional experiments in order to better understand the different mechanisms behind Re-Tuning.

7.2.1 Ablation Study

For all tasks, as the problem size grows, so does the number of unique possible problems. For example, there are more combinations of 10-digit addition problems then there are 2-digit addition problems. If we randomly sample problems from the space of all possible problems up to some length, then the distribution of problems will be skewed towards longer problem instances. Due to Re-Tuning's recursive design, it's important that an appropriate number of small problems are included in the training data. Our resampling methodology is described in Appendix 9.3.1

To better understand the impact of resampling short problem instances for finetuning, we finetune LLaMA-7b using integer addition data with and without resampling. Both models saw the same number of training examples. We collect results on 100 problems of length 5, 20, 35, and 50. Results are shown in Figure 7.1.

	5	20	35	50
w/ Resampling	1.0	0.98	0.96	0.69
w/o Resampling	1.0	0.97	0.73	0.0

Table 7.1: Ablation over resampling approach during Re-Tuning finetuning.

While both trained models perform well on problems up to length 20, the superiority of the resampling approach becomes clear on longer problems. At a problem length of 35, the resampling model achieved 96% accuracy, while the accuracy of the model trained without resampling is only 73%. At a problem length of 50, the model trained without resampling fails to correctly solve a single problem instance. However, the model trained with resampling maintains an accuracy of 69%, which suggests that resampling shorter problem instances is important for the success of Re-Tuning.

7.2.2 Case Study

The scratchpad models often make errors with indexing operations. For example, on the dynamic programming problem, the largest training array size is 5 elements. In Figure 7.2 we see that the model generated scratchpad starts the index of the array of sub-array sums (dp

Prompt

```
Question: Let's solve input = [1, 0, -2, 1, 0, 4, 2].
```

Scratchpad Model Response

```
Scratchpad: dp[5] = max(input[5], 0) = max(4, 0) = 4

dp[4] = max(input[4], input[5], 0) = max(1, 4, 0) = 4

dp[3] = max(dp[4], input[3] + dp[5], 0) = max(4, -2 + 4, 0) = 4 ...

[rest of output truncated]
```

Re-Tuning Response

```
Call: Create dp array [1, 0, -2, 1, 0, 4, 2]\nReturn: [6, 5, 5, 5, 4, 4, 2]

Answer: Call: Create chosen indices array:

sum array [6, 5, 5, 5, 4, 4, 2], item array [1, 0...

Return: [1, 2, 2, 1, 2, 1, 2]\nAnswer: [1, 2, 2, 1, 2, 1, 2]
```

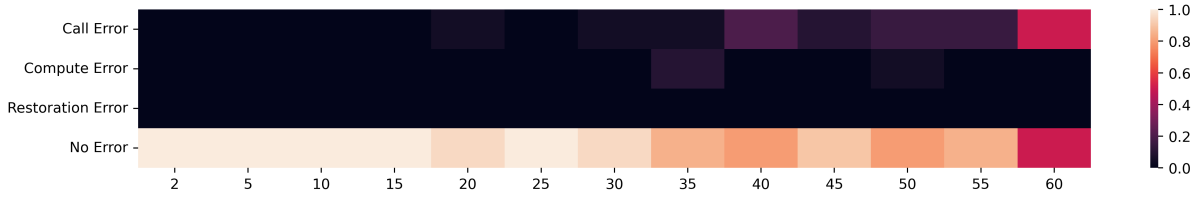
Figure 7.2.: Dynamic Programming Example: Prompt and Responses (Truncated)

array) at 5 given an input array of length 7. The dp array should start at the last element of a 7 element array, with index 6 instead of 5. Once the model makes this indexing error on the scratchpad it is unable to recover. In other examples not displayed here, the scratchpad method correctly generates the text for "dp[6]" but fails to populate the subsequent expressions with the correct values from the input array. It is unable to reliably extract the elements with the correct index from the input array. In contrast, the Re-Tuning method is shown in Figure 7.2. Only the initial context is shown to save space. The Re-Tuning model is able to generate prompts with no indexing difficulty in this case.

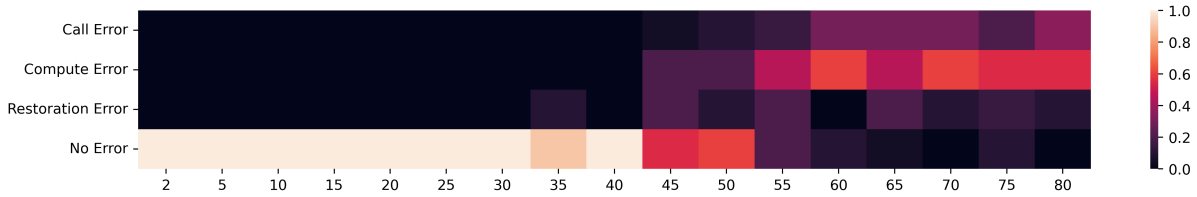
7.2.3 Error Analysis

In order to better understand the types of errors made by Re-Tuning models, we perform extensive error analysis on each task. For each problem, we sample 20 problem instances from a selection of problem lengths, and categorize them into the following types:

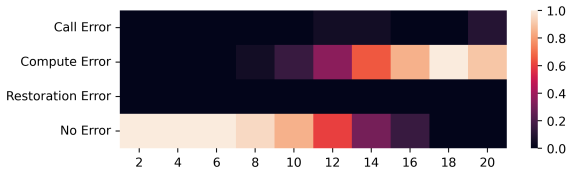
- Call Error: At some point in the call stack, an incorrect recursive call is made. As a result, the input prompt to the new context is incorrect.
- Compute Error: This error can manifest either because the base case is incorrectly solved, or at some point in the call stack the models returns the wrong solution to a subproblem even though the correct answer to its recursive call was received. As a result, the answer returned by the current context to the earlier context will be incorrect.
- Restoration Error: A restoration error occurs if at some point in the call stack, a call error or compute error is made, yet later recovered such that the final answer to the prompt in



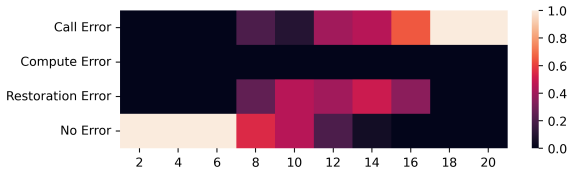
(a) Errors on Integer Addition



(b) Errors on Parity Problem



(c) Errors on Dynamic Programming (subproblem 1)



(d) Errors on Dynamic Programming (subproblem 2)

Figure 7.3.: Error classifications for each problem across samples of size 20 for a selection of problem lengths.

the initial context is correct. Importantly, since the model is able to recover, instances of restoration errors are classified as correct.

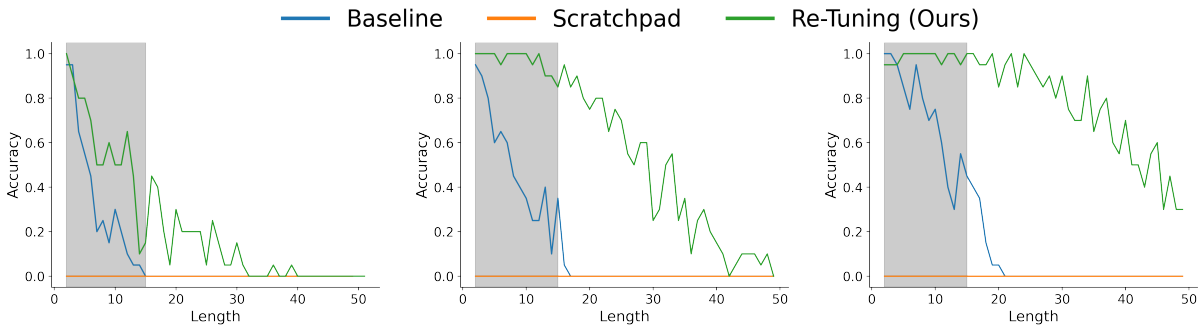
- No Error: In order for a problem instance to be free of errors, each recursive call must be correct, the base case must be solved correctly, and the correct answers are propagated up the call stack, leading to the correct final answer.

Figure 7.3 displays the error classifications for each problem. Importantly, across all tasks, the prevalence of errors increases with the size of the problem.

On the integer addition task, very few call errors and compute errors occur before a problem size of 30. The vast majority of those errors are call errors, suggesting that the model has a difficult time constructing the subproblem. Importantly, this aligns with [46], which suggest that simply copying long strings of text with repeating characters is a difficult task for transformer-based models to perform. Furthermore, the lack of restoration errors suggest that once a call error is made, the model has a very hard time recovering.

For the parity problem, we again see very few errors of any type before a problem size of 40. In contrast with the addition problem, the majority of errors made on the parity problem are compute errors, not call errors. This is rather unintuitive, as the addition operation is seemingly much harder than the possible polarity flip in the polarity problem. Interestingly, we see more cases of restoration errors in the parity problem, which suggests that it may be easier for the model to recover from call or compute errors on this task compared to integer addition.

For the dynamic programming problem, we perform error analysis on each subproblem separately. The first subproblem deals with constructing the an array of sub-array sums, while the section subproblem identifies which indices correspond to the maximum sum. While compute errors occur most frequently on the first subproblem, call and restoration errors occur more frequently on the second subproblem. This checks out, as the first subproblem requires a rather simple call, but involves a more complex step to compute the answer, whereas the second subproblem contains a more involved recursive call, but an easier computation to produce the index array. Furthermore, the prevalence of restoration errors on subproblem 2 suggest that these call errors are easier to recover from than the computer errors made in subproblem 1.



(a) 10 Examples per Length (b) 25 Examples per Length (c) 50 Examples per Length

Figure 7.4.: Results on Addition with limited training data of 10 examples per length (left), 25 examples per length (middle), and 50 examples per length (right). Scratchpad results are omitted since they were constant at 0% accuracy.

7.2.4 Sample Efficiency

We also run experiments to see the performance of Re-Tuning in the low-data regime on addition. The results are in Figure 7.4. For each experiment we construct training data consisting of n examples for each problem length, where n is 10, 25, and 50 and the numbers contain between 1 and 15 digits. For example, when n is 10, the model will see 10 examples of adding 2 1-digit numbers, 10 examples of adding 2 2-digit numbers, etc. So it sees 150 examples in total when n is 10. We train baseline, scratchpad, and Re-Tuning models on these examples for 5 epochs each. We do not use any resampling as in the other experiments.

With seeing only 50 examples per problem length, the Re-Tuning model has performance close to the Re-Tuning model in the full-data regime above. In contrast, the baseline model has much worse performance than the baseline model in the full-data regime. With only seeing 10

examples per problem length in training, the Re-Tuning model is comparable to the baseline model that sees 50 examples per problem length in training, a 5x efficiency increase.

7.2.5 Prompt Sensitivity

Here, we explore the sensitivity of Re-Tuning models to various prompts during inference. Specifically, we take our best LLaMA-7b trained on the integer addition problem with Re-Tuning using the prompt "`{num_1} + {num_2}\nSolution: `", and we evaluate the model using several alternative prompt formats for inference. Results are shown in Table 7.2 on 100 problems of length 5, 20, 35, and 50.

	5	20	35	50
"`{num_1} + {num_2}\nSolution: `"	1.0	0.98	0.96	0.69
"`{num_1} + {num_2}\nAnswer: `"	1.0	1.0	0.86	0.65
"`{num_1} + {num_2}\n `"	1.0	0.98	0.88	0.67
"`{num_1} - {num_2}\nSolution: `"	0.28	0.26	0.18	0.07

Table 7.2: Prompt sensitivity analysis on LLaMA 7B with Re-Tuning on the integer addition problem.

The results suggest that during inference, Re-Tuning is not very sensitive to minor deviations in the prompt. The first prompt in Table 7.2 is the prompt used during training. The second and third prompts include small prompt deviations, and result in slightly worse performance on longer problems. Specifically, the 2nd prompt uses "`Answer: `" in an attempt to have the model skip the recursive call, but it appears that Re-Tuning is robust against such attacks. Re-Tuning however, is not robust against the fourth prompt, which adversarially prompts for subtraction rather than addition, resulting in significantly worse performance.

8. Related Work and Future Directions

8.1 Related Work

Several works have explored the ability of LLMs for length generalization on compositional problems. [36] suggests that LLMs solve compositional tasks via "linearized subgraph matching" and thus fail to learn the underlying algorithm necessary to solve more complex problem instances. [39] showed that finetuning on a combination of in-context learning and scratchpad prompting could enable better performance. Similarly Re-Tuning involves finetuning pretrained LLMs to make recursive calls in order to improve performance on composition tasks. Other works have studied length generalization on small, purpose-built transformer models. [50] and [46] showed that training small transformers models from scratch on scratchpad data could enable better length generalization.

Various papers have explored the idea of LLMs prompting themselves or other LLMs, although no papers to our knowledge that explicitly finetune a language model to do so. [51] prompts a language model to break a problem down into simpler steps and then prompts itself to solve each step individually in a sequential, non-recursive manner. Similar methods have been proposed as way to improve the logical consistency of the generated repsonses [52, 53].

[54] use a language model to generate prompts by extracting relevant information from the context. They call this "System 2 Attention". The idea behind this also underlies Re-Tuning. In Re-Tuning, the language model does 2 operations in each context: it generates a prompt for a subproblem (for example a prompt to add smaller numbers than the input) and does some local computations (for example add 2 digits in a number and combine that with the previously computed sum of the other digits). The prompt for the subproblem removes irrelevant information for solving the subproblem. In the addition example, to add the 10th digits of 2 numbers, the 11th digits of the numbers are irrelevant. These larger digits are removed in the prompt for the subproblem, similar to how [54] remove irrelevant information in a prompt.

A recent topic of interest has been teaching language models to use tools [47, 48, 55–58], which often involve stopping the generation and waiting for the tool output before containing with generation. Re-Tuning can be interpreted as teaching LLMs to use themselves as a tool. We will discuss this interpretation more in the next section.

Several papers have investigated the ability of language models on arithmetic tasks [37, 38, 50, 59–61]. In many cases, it was noticed that performance was significantly worse on problems longer than those saw during finetuning.

8.2 Future Directions

While the experimental results of the Re-Tuning are strong compared to other methods, the tasks we evaluated on are relatively simple. For each task we already know an algorithm to solve the problem. We train the model to follow this algorithm exactly by creating finetuning data with the steps of the algorithm written out. Our training data explicitly tells the model what

subproblems it should generate a prompt for and how to use the solutions to these to compute the final answer. A natural question is: can Re-Tuning be useful for tasks that we don't have an explicit method for solving? For example, could a language model using Re-Tuning respond to a general user query by breaking down its solution process into various subproblems, generate prompts for those, then use all the answers to these subproblems to return a final answer to the user?

One method to proceed in this direction is to repurpose methods that teach language models to use tools, for example [47]. These types of methods train language models to use tools without needing large amounts of training examples showing how to use the tools. Instead, the language models generate examples of using tools, which are then filtered according to how good they are. The language model is then fine-tuned on this new tool-use data. The same type of method could be applied for teaching a language model to prompt itself, which can be thought of as the language model using itself (or possibly another model) as a tool.

Re-Tuning could also help reduce the context size. While current state-of-the-art models have contexts of 100k tokens or even millions of tokens, they are not able to effectively reason over the entire context very well [62]. Rather than processing all the information in one context, Re-Tuning allows the language model to process different parts of a problem in different contexts, which could alleviate problems associated with large contexts. For example, in 6.1, we see that Re-Tuning does not tend to have much longer contexts than the baseline method, despite generating many more tokens in total than the baseline method.

9. Appendix

9.1 Glossary of main symbols for reaction-diffusion chapters

Symbol	description	defined in chapter
$\mathcal{A}_\epsilon(n,b)$, $A_\epsilon(n,b)$	coss-section of b and its area at juncture n	3.4
$\mathcal{A} = \mathcal{L}^*$	differential operator $\mathcal{A}$, the adjoint of $\mathcal{L}$	5.1
$b, \mathbf{b} = (n, n'), \overline{\mathbf{b}}$	branch, oriented branche, reverse orientation	3.2
$c_i(\mathbf{x}, t)$	concentration of i at $\mathbf{x}$ in $\mathcal{R}$ at time t	3.1
$D_i(\mathbf{x})$	diffusion coefficient of i at $\mathbf{x}$ in $\mathcal{R}$	3.1
$\deg(n)$	degree of node n	3.2
$f_{ij}(n)$	output composition matrix	3.1
$\mathbf{j}_i(\mathbf{x})$	flux density of i at $\mathbf{x}$	3.1
$\kappa_{ij}(\mathbf{x})$, $K_{ij}(n)$, $\tilde{K}_{ij}(n)$	reaction coefficients in reaction domain and on network	3.2, 3.4, 3.8
$\ell(b)$, $\tilde{\ell}(\mathbf{b})$	length of branch, velocity-adjusted length	3.2, 3.7
$\mathcal{L}$	differential operator of reaction-transport equation	5.1
n	network node	3.2
$\mathbf{n}(\mathbf{x})$	normal vector field to boundary of $\mathcal{R}$	3.1
$p(n,b)$	cross-section area fraction of b at juncture n	3.2, 3.4
$\mathcal{R}$	network-like reactor domain, network reactor	3.1
$\mathbf{v}_i(\mathbf{x})$, $\nu_i(\mathbf{b})$, ν_i^b	advection velocity of i	3.1, 3.2
$\eta_i(n,b)$	coefficients of matrix Λ	3.8, 4
$\varphi_{\mathbf{b}}(x)$	arc-length parametrization of $\mathbf{b}$	3.2
$\Lambda(n_i, n_j)$, λ	coefficient matrix and vector in linear algebraic equations for f	3.8
$\xi_i(n,b)$	branch coefficients in definition of $\eta_i(n,b)$	3.8, 4
$\rho(x,t\|y,s)$	fundamental solution of reaction-transport equation	5.1

9.2 Finetuning Details

Finetuning (and evaluation) were done on NVIDIA H100, A100, and RTX A6000 GPUs, depending on availability and the compute requirements of the job. Rather than finetune the full model, we finetune using low-rank adapters [49]. Hyperparameters for finetuning jobs are in Table 9.1. These hyperparameters apply to finetuning all models across all tasks, with two notable exceptions: (1) when training parity baselines, we used a slightly higher learning rate of 5e-4 for better stability, and (2) the scratchpad finetuning job for the dynamic programming problem on LLaMA 13B used a batch size of 64, along with 64 gradient accumulation steps, so that the training job could be done on a single A100 GPU.

Parameter	**Value**
Learning Rate	2×10^{-4}
LR Schedule	Constant
Optimizer	AdamW
Batch Size	128
Gradient accumulation steps	32
Lora_r	64
Lora_alpha	64
Lora_dropout	0.05

Table 9.1: Hyperparameters used for finetuning.

During training, checkpoints are saved every 7936 training examples. We evaluate a handful of these checkpoints on a small validation set containing 5 examples from a handful of problem

lengths (both in-distribution and out-of-distribution) in order choose the best model for full evaluation. We compute training loss (using cross entropy loss) only on the parts of the problem that the model will generate at inference time (c.f. grey vs. green highlighted text in the upper right of Figure 6.1).

9.3 Data Details

9.3.1 Resampling methodology

In general, we upsample examples with smaller lengths and downsample those with larger lengths in our training data. There are 2 reasons for this. One is that the examples with larger lengths are more numerous than examples with smaller lengths (there are many more examples of adding 2 15-digit numbers than there are adding 2 1-digit numbers). The other reason is specific to Re-Tuning. Since Re-Tuning generates calls to all examples except the base case, it has trouble learning the what to do in the base case if there are not enough examples. Without resampling, the base cases for each problem would be far less than 1% of the training data. We do not follow any specific methodology for resampling. We simply try to bring the training data distribution closer to uniform than it would be without resampling. See Figure 9.1 for a comparison of training data on the dynamic programming problem before and after resampling. Note the y-axis scales on each plot. Without resampling, examples of lengths 1 and 2 make up far too little of the data distribution for the model to learn them. We apply similar resampling in the other problem settings. See Figure 9.2 for a comparison of training data on the addition problem before and after resampling. See Figure 9.3 for a comparison of training data on the parity problem before and after resampling.

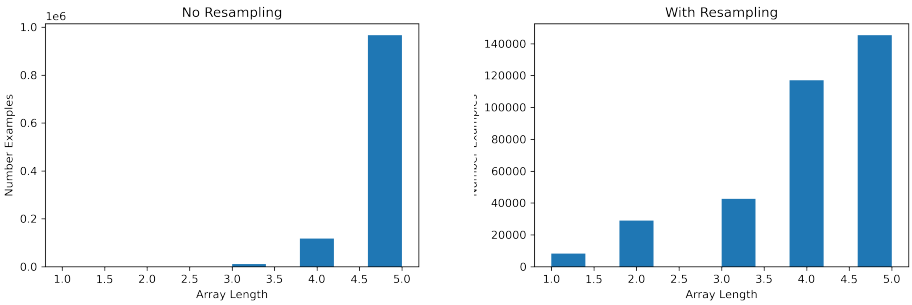


Figure 9.1.: Dynamic Programming Training Data Resampling Comparison

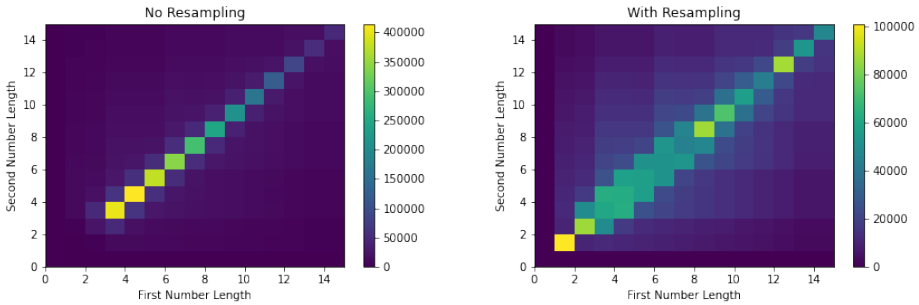


Figure 9.2.: Addition Training Data Resampling Comparison

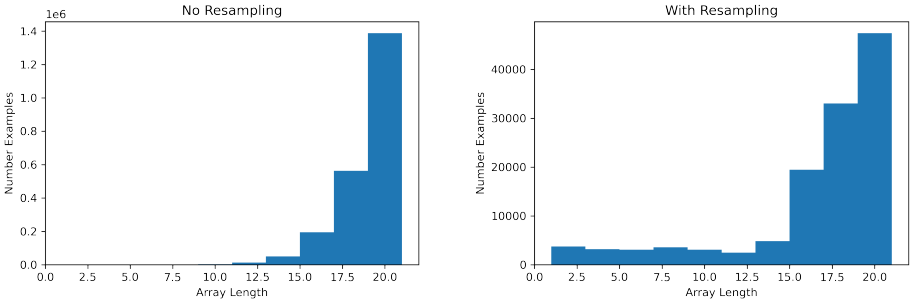


Figure 9.3.: Parity Training Data Resampling Comparison

9.4 Psuedo-code for Re-Tuning generation

Psuedo-code for the recursive generation using in Re-Tuning is shown in Figure 9.4.

```
def recursive_generate(model, prompt):
    context = model.generate(prompt)

    # loop until all calls executed
    while exists_unexecuted_call(context):
        call = get_latest_call(context) # get latest call
        if correct_format(call):
            retrn = recursive_generate(model, call)
        else: Raise Exception

        call_answer = extract_answer(retrn) # get final answer from context
        new_prompt = context + call_answer # append answer from recursive context
        context = model.generate(new_prompt) # continue generation

    return context
```

Figure 9.4.: Pseudo-code for Re-Tuning generation.

9.5 Problem Examples

9.5.1 Re-Tuning Addition

The original addition data from [38] consists of numbers of up to 15 digits. For this we have the model add each digit individually. The model outputs the current sum and the current carry separately. For adding 637 and 123, the finetuning data is

```
'637 + 123\nSolution:
Call: 37 + 23\n
Return: Carry 0, Output 60\n
Answer: Carry 0, Output 760'

'37 + 23\nSolution:
Call: 7 + 3\n
Return: Carry 1, Output 0\n
Answer: Carry 0, Output 60'

'7 + 3\nSolution:
Carry 1, Output 0'
```

To compute the accuracy at generation, we extract the carry and output from the initial context. If the carry is 1, we prepend it to the output, else we leave the output as is. Then we cast the output sum to an integer and compare with the true sum to get the accuracy.

9.5.2 Re-Tuning Dynamic Programming

The initial data for this task consists of arrays of up to 5 numbers and each number is an integer from -5 to 5.

The finetuning data for the dynamic programming problem consists of 3 types of data: initial context, dp array contexts, and indices array contexts. In the initial context, the model creates prompts to create the dp array and the indices array (the indices array is the final output). The dp array contexts solve the subproblem to create an array with the sub-array sums and the indices array contexts solve the subproblem to create the final solution given the dp array. Examples of each of these training examples are shown below:

Listing 9.1: Initial Context Examples

```
'Compute the maximum sum of nonadjacent subsequences of [1, -1, 3]
\nSolution:
Call: Create dp array [1, -1, 3]\n
Return: [4, 3, 3]\n
Answer: Call: Create chosen indices array:
sum array [4, 3, 3], item array [1, -1, 3], can use item True\n'


'Compute the maximum sum of nonadjacent subsequences of [3, 2]
\nSolution:
Call: Create dp array [3, 2]
\nReturn: [3, 2]\n
Answer: Call: Create chosen indices array:
```

```
sum array [3, 2], item array [3, 2], can use item True\n'
```

Listing 9.2: DP Array Context Examples

```
'Create dp array [1, -1, 3]\n
Solution: Call with array minus first element.
Call: Create dp array [-1, 3]\nReturn: [3, 3]\n
Answer: Append max(return[0], array[0] + return[1])
to return.\nAnswer: [4, 3, 3]'

'Create dp array [-1, 3]\n
Solution: Call with array minus first element.
Call: Create dp array [3]\nReturn: [3]\nAnswer:
Append max(return[0],array[0] + return[1]) to
return.\nAnswer: [3, 3]'

'Create dp array [3]\n
Solution: Return [0] if negative else element.\n
Answer: [3]'
```

Listing 9.3: Indices Array Context Examples

```
'Create chosen indices array: sum array [4, 3, 3], item array
[1, -1, 3], can use item True\n
Solution: If there is only 1 item, return 1 if we should
```

```
use it else 2. If we should use the first item to get the sum,
call False else True.
Call: Create chosen indices array: sum array
[3, 3], item array [-1, 3], can use item False
\nReturn [2, 1]\nAnswer:
Append 1 if False else 2.\nAnswer: [1, 2, 1]'

'Create chosen indices array: sum array [3, 3], item array
[-1, 3], can use item False\n
Solution: If there is only 1 item, return 1 if we should
use it else 2. If we should use the first item to get the
sum, call False else True. Call: Create chosen indices array: sum
array [3], item array [3], can use item True\n
Return [1]\nAnswer: Append 1 if False else 2.\n
Answer: [2, 1]'

'Create chosen indices array: sum array [3], item array [3],
can use item True\nSolution: If there is only 1 item, return
1 if we should use it else 2.\nAnswer: [1]'
```

To compute the accuracy for the dynamic programming problem, we parse out the index array in the initial context and compare to the string of the true solution array.

9.5.3 Re-Tuning Parity

For the parity problem the model computes the parity of the binary array in the prompt given the parity of the sub-list of the last n-1 items. So the finetuning examples for the parity of `[1, 0, 0, 1]` are:

```
'What is the parity of [1, 0, 0, 1]?\n
Solution: Call: What is the parity of
[0, 0, 1]?\n\n
Return: 1\nAnswer: 0'


'What is the parity of [0, 0, 1]?\n
Solution: Call: What is the parity of
[0, 1]?\n\n
Return: 1\nAnswer: 1'


'What is the parity of [0, 1]?\n
Solution:
Call: What is the parity of [1]?\n\n
Return: 1\nAnswer: 1'


'What is the parity of [1]?\n
Solution: 1'
```

To evaluate accuracy, when the model has finished we simply take the last (non-empty) element of the resulting string, and compare that to the true answer (as a string).

9.5.4 Scratchpad Addition

We have the model learn a scratchpad to add the digits in the 2 numbers individually. The data is very similar to the recursive addition data except there are no generated prompts and all the computations are in 1 context. To add 3116 + 5923, the data would be:

```
'3116 + 5923\nSolution:
Carry 0, Output 9\nCarry 0, Output 39\nCarry 1,
Output 039\nCarry 0, Output 9039'
```

Similar to the recursive finetuning, we do not compute training loss on the prompt, which would be `'3116 + 5923\nSolution: '` in this example. We compute the accuracy in the same way as with the recursive addition.

9.5.5 Scratchpad Dynamic Programming

For this we use the same scratchpad as [36]. So to solve the problem for the array [3, -2, 2], the training example is

```
'Question: Let's solve input = {arr}.
Scratchpad: dp[2] = max(input[2], 0) =
max(2, 0) = 2
dp[1] = max(input[1], input[2], 0)
= max(-2, 2, 0) = 2
```

```
dp[0] = max(dp[1], input[0]+ dp[2], 0) =
max(2, 3 + 2, 0) = 5

Finally, we reconstruct the lexicographically
smallest subsequence that fulfills the task objective
by selecting numbers as follows. We store the
result on a list named "output".

Let can_use_next_item = True. Since dp[0] == input[0] +
dp[2] (5 == 3 + 2) and can_use_next_item == True,
we store output[0] = 1. We update can_use_next_item
= False. Since dp[1] != input[1] (2 != -2)
or can_use_next_item == False, we store output[1] = 2. We
update can_use_next_item = True. Since dp[2] == input[2] (2 == 2)
and can_use_next_item == True, we store output[2] = 1.

Reconstructing all together, output=[1, 2, 1].'
```

To compute the accuracy, we parse out the index array and compare this string to the string of the true answer. This is the same as with the recursive dynamic programming method.

9.5.6 Scratchpad Parity

The original data consists of arrays up to length 21 with entries that are either 0 or 1. We construct the finetuning data in the following way.

We have the model learn to sequentially sum the elements of the input array modulo 2, similar to the recursive method. However all the computations are done in 1 context with no recursive calls. The scratchpad we use is similar to that of [39]. For the input [1, 0, 1] the finetuning data is:

```
'What is the parity of [1, 0, 1]?
\nSolution: Compute one element at a time. 1 1 0'
```

The last number in the generation is the final answer. We compute the accuracy exactly the same as in the recursive case.

9.5.7 Baseline Addition

The baseline addition data consists of prompts to add 2 numbers. The target is just the sum of the numbers.

```
'24 + 97\nAnswer: 121'
```

9.5.8 Baseline Dynamic Programming

The prompt for baseline dynamic programming data and the response are below:

```
'Given a sequence of integers, find a subsequence with the
highest sum, such that no two numbers in the subsequence
```

```
are adjacent in the original sequence.\n\nOutput a list
with "1" for chosen numbers and "2" for unchosen ones. If
multiple solutions exist, select the lexicographically smallest.
Input = [1, -3, 2].\nAnswer: [1, 2, 1]'
```

9.5.9 Baseline Parity

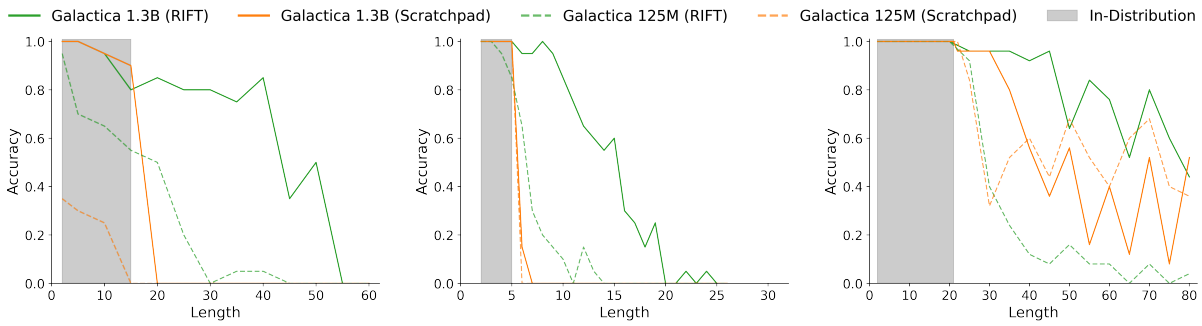
The prompt for baseline parity and the response are below:

```
What is the parity of [0, 1, 0]?\nAnswer: 1
```

9.6 Additional Results

In this section, we share Scratchpad and Re-Tuning results on two additional models: Galactica 1.3B and Galactica 125M [41]. Results across our 3 tasks are shown in Figure 9.5.

On the addition task, both Scratchpad and Re-Tuning methods on Galactica 1.3B produce highly accurate results up to length 15, which is the maximum problem size seen during training. While the scratchpad accuracy quickly drops to 0% for problems beyond length 15, Re-Tuning is accuracy hovers near 80% through a problem length of 40, and only reaches 0% accuracy for problems at length 55. Certainly, the performance of both methods is worse for Galactica 125M, but Re-Tuning still performs much better than Scratchpad prompting. With Scratchpad, accuracy never tops 40%, even for lengths seen during training, and quickly falls to 0% at length 15. However, with Re-Tuning, Galactica 125M is able to achieve greater than 50% accuracy even at length 20.



(a) Integer Addition (b) Dynamic Programming (c) Parity

Figure 9.5.: Scratchpad and Re-Tuning results on Addition (left), Dynamic Programming (middle), and Parity (right) on Galactica 1.3B ad Galactica 125M. The span of problem lengths seen during training are highlighted in grey for each problem.

Results on the dynamic programming problem are quite similar, with Re-Tuning able to produce near 50% accuracy on problems of length 15 when using Galactica 1.3B, and near 50% accuracy on length 7 using Galactica 125M. With Scratchpad prompting, both models achieve perfect in-distribution performance, but accuracy quickly falls to 0% on problems of length 6 and 7 for Galactica 125M and 1.3B respectively. Interestingly, while Galactica 1.3B with scratchpad fails to solve a single problem of length 7 correctly, Galactica 125M with Re-Tuning can still solve near 50% correctly, even at just 1/10th of the size.

Results on parity are more interesting. Though the best performance overall is achieved by Galactica 1.3B with Re-Tuning, the worst overall performance is achieved by Galactica 125M with Re-Tuning. With Scratchpad prompting, both models perform similarly, with Galactica 1.3B performing better on shorter problems, while Galactica 125M performs better on longer problems. The parity problem is interesting because the random performance should be near

50% given that there are only two options, 0 or 1, which is around the performance we see on long problem lengths for both models when using scratchpads. Galactica 125M with Re-Tuning, however, performs worse than random for problem lengths beyond 30. Looking closer, almost all the errors made by Galactica 125M with Re-Tuning on this task are call errors (see Section 7.2.3 for a description of each error categorization), suggesting that Galactica 125M has a particularly difficult time decreasing the problem for the recursive call, which involves copying all but the first element of the binary array. In the scratchpad setup, however, models attempt to keep a running counter of the parity as as they traverse through the binary array, only outputting 0's or 1's in the process. As a result, the scratchpad setup converges around 50%, while the accuracy with Re-Tuning can fall below random chance as a result of errors made in formulating subsequent calls.

REFERENCES

[1] Renato Feres, Eric Pasewark, and Gregory Yablonsky. Determination of output composition in reaction-advection-diffusion systems on network reactors, 2024.

[2] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[3] Xiaowu Dai and Yuhua Zhu. On large batch training and sharp minima: A fokker-planck perspective, 2021.

[4] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

[5] R. Feres, M. Wallace, A. Stern, and G. Yablonski. Explicit formulas for reaction probability in reaction-diffusion experiments. *Computers & Chemical Engineering*, 125:612–622, 2019.

[6] R. Feres, M. Wallace, and G. Yablonsky. Reaction-diffusion on metric graphs: from 3d to 1d. *Computers & Mathematics with Applications*, 73(9):2035–2052, 2017.

[7] R. Aris. *The Mathematical Theory of Diffusion and Reaction in Permeable Catalysts, Vol. I.* Clarendon Press - Oxford, 1975.

[8] D.A. Frank-Kamenetskii. *Diffusion and heat exchange in chemical kinetics*. Princeton Legacy Library, 1955.

[9] G.S. Yablonskii, V.I. Bykov, A.N. Gorban, and V.I. Elokhin. *Kinetic Models of Catalytic Reactions*, volume 32 of *Comprehensive Chemical Kinetics*. Elsevier, 1991.

[10] D. Constales, G.S. Yablonsky, G.B. Marin, and J.T. Gleaves. Multi-zone tap-reactors: theory and application. 1. the global transfer matrix expression. *Chem. Eng. Sci.*, 56:133–149, 2001.

[11] D. Constales, G.S. Yablonsky, G.B Marin, and J.T. Gleaves. Multi-zone tap-reactors, theory and application ii the three dimensional theory. *Chem. Eng. Sci.*, 56:1913–1923, 2001.

[12] D. Constales, G.S. Yablonsky, G.B. Marin, and J.T. Gleaves. Multi-zone tap-reactors, theory and application, iii. multi–response theory and criteria of instantaneousness. *Chem. Eng. Sci.*, 59:3725–3736, 2004.

[13] D. Constales, S.O. Shekhtman, G.S. Yablonsky, G.B. Marin, and J.T. Gleaves. Multi-zone tap-reactors, theory and application, iv. the ideal and non-ideal boundary conditions. *Chem. Eng. Sci*, 61:1878–1891, 2006.

[14] Z. Gromotka, G. Yablonsky, N. Ostrovskii, and D. Constales. Integral characteristic of complex catalytic reaction accompanied by deactivation. *Catalysts*, 12:1283, 2022.

[15] P.V. Danckwerts. Continuous flow systems. *Chemical Engineering Science*, 2:1–13, 1953.

[16] P.V. Danckwerts. The effect of incomplete mixing on homogeneous reactions. *Chemical Engineering Science*, 8:93–99, 1958.

[17] Th.N. Zwietering. The degree of mixing in continuous flow systems. *Chemical Engineering Science*, 11:1–15, 1959.

[18] J.T. Gleaves, G.S. Yablonskii, P. Phanawadee, and Y. Schuurman. Tap-2. interrogative kinetics approach. *Applied Catalysis A: General*, 160:55–88, 1997.

[19] G.B. Marin, G. Yablonsky, and D. Constales. *Kinetics of Chemical Reactions: Decoding Complexity*. Wiley-VCH, 2 edition, 2019.

[20] Y. Wang, M.R. Kunz, G.S. Yablonsky, and R. Fushimi. Accumulation dynamics as a new tool for catalyst discrimination: An example from ammonia decomposition. *Industrial & Engineering Chemistry Research*, 58(24):10238–10248, 2019.

[21] G.S. Yablonskii, S.O. Shekhtman, S. Chen, and J.T. Gleaves. Moment-based analysis of transient response catalytic studies (tap experiment). *Industrial & Engineering Chemistry Research*, 37:2193–2202, 1998.

[22] V. Volpert. *Elliptic Partial Differential Equations Vol. 2: Reaction-Diffusion Equations*, volume 104 of *Monographs in Mathematics*. Birkhäuser, 2014.

[23] A. Friedman. *Partial Differential Equations of Parabolic Type*. Robert E. Krieger Publishing Company, 1983.

[24] G.S. Yablonsky, D. Constales, and G.B. Marin. Joint kinetics: a new paradigm for chemical kinetics and chemical engineering. *Current Opinion in Chemical Engineering*, 29:83–88, 2020.

[25] G.S. Yablonsky, D. Branco, G.B. Marin, and D. Constales. New invariant expressions in chemical kinetics. *Entropy*, 22:373, 2020.

[26] S.D. Eidel'man. *Parabolic Systems*. North-Holland, 1969.

[27] D. Hoff. *Linear and Quasilinear Parabolic Systems: Sobolev Space Theory*. Mathematical Surveys and Monographs, V. 251, AMS, 2020.

[28] A. Amosov and G. Pansenko. Partial dimension reduction of a domain containing thin tubes for solving the diffusion equation. *Journal of Mathematical Sciences*, 264(5), July 2022.

[29] S. Albeverio and S. Kusuoka. Diffusion processes in thin tubes and their limits on graphs. *The Annals of Probability*, 40(5):2131–2167, 2012.

[30] M. Freidlin and S.-J. Sheu. Diffusion processes on graphs: stochastic differential equations, large deviation principle. *Probab. Theory Relat. Fields*, 116:181–220, 2000.

[31] Hannes Risken. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer Series in Synergetics. Springer Berlin, Heidelberg, 1996.

[32] Grigorios A. Pavliotis. *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*. Texts in Applied Mathematics. Springer New York, NY, 2014.

[33] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, 2013.

[34] OpenAI. GPT-4 Technical Report. *arXiv*, 2023.

[35] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[36] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023.

[37] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

[38] Tiedong Liu and Bryan Kian Hsiang Low. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks, 2023.

[39] Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[40] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

[41] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science, 2022.

[42] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas

Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *arXiv*, 2022.

[43] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang,

Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. PaLM 2 Technical Report. *arXiv*, 2023.

[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[45] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv*, 2023.

[46] Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization, 2023.

[47] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[48] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023.

[49] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[50] Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers, 2023.

[51] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[52] Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations, 2022.

[53] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models, 2023.

[54] Jason Weston and Sainbayar Sukhbaatar. System 2 attention (is something you might need too), 2023.

[55] Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models, 2023.

[56] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models, 2022.

[57] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

[58] Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023. Survey Certification.

[59] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks, 2021.

[60] Jeonghwan Kim, Giwon Hong, Kyung-min Kim, Junmo Kang, and Sung-Hyon Myaeng. Have you seen that number? investigating extrapolation in question answering models. In

Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7031–7037, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[61] Shaoxiong Duan and Yining Shi. From interpolation to extrapolation: Complete length generalization for arithmetic transformers, 2023.

[62] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.